Products Rights Notice:

You understand that these materials were not prepared for public release and you assume all risks in using these materials. These risks include, but are not limited to errors, inaccuracies, incompleteness and the possibility that these materials infringe or misappropriate the intellectual property right of others. You agree to assume all such risks.

THESE MATERIALS ARE PROVIDED BY THE COPYRIGHT HOLDERS AND OTHER CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS (INCLUDING ANY OF OWNER'S PARTNERS, VENDORS AND LICENSORS) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THESE MATERIALS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun, Sun Microsystems, the Sun logo, Solaris, OpenSPARC T1, OpenSPARC T2 and UltraSPARC are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. The Adobe logo is a registered trademark of Adobe Systems, Incorporated. Part of the products covered by these materials may be derived from the Berkeley BSD systems licensed by the University of California. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product described in these materials. This distribution may include materials developed by third parties who have intellectual property rights therein. Products covered by and information contained in these materials may be controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists may be prohibited.

Table of Contents

LIST OF DIAGRAM

List of Tables

Chapter 1 microSPARC-II Overview

1.1 Introduction

The microSPARC-II CPU is an highly integrated, low cost implementation of the SPARC RISC architecture, evolving from SUN's microSPARC architecture. High performance is achieved by the high level of integration including on chip instruction and data caches and the close coupling of the CPU with main memory. A full custom implementation allows for a minimum target frequency of 70MHz providing sustained performance of 40+ Specmarks with SC1.0. Sparc compilers and no preprocessing. Estimate Specmarks with SC3.0 is 48. The design is highly testable with the use of the full JTAG scan support. The microSPARC-II chip will support up to 256MB of DRAM and 5 SBus slots.

1.2 Key differences between microSPARC-II and microSPARC

1.2.1 Overall

- Virtual index virtual tag scheme is used for the cache RAMs to minimize the critical path for tag match. This changes the MMU and cache design.
- Single cycle integer store and 2-cycle integer store double via new dedicated store read port in the register file.
- All floating point loads and stores, including double word operations are single cycle operations
- 3.3 volt operation is used for internal logic. 5 volt I/O is used for SBus interface and 3volt/5volt is used for other pin I/O for low power consumption. See microSPARC-II Datasheet for details.
- Phase lock loop is used to generate the internal high frequency clock.

1.2.2 IU:

- Integer branches are folded with their delay instruction. Static branch prediction is used.
- Incorporate a four-entry instruction queue with prefetch.
- Implement Instruction and Data cache streaming extensions.
- Instruction access exceptions are detected in IU.

- Implement a 8-window SPARC register file.

1.2.3 FPU:

- A multiplier array is incorporated in addition to the Meiko core. Back-to-back Fmuls/Fmuld takes 3 cycles, Fmuls/Fmuld followed by dependent store take 4 cycles.
- FPU has 3 queue entries for improved performance.

1.2.4 Instruction Cache:

- 16 K byte direct map cache with 32 byte block size. Virtual address and context are used in the tag along with protection check.
- No flash clear. Standard Sparc V8 flush instruction is used to clear one entry of the cache.

1.2.5 Data Cache:

- 8 K byte direct map cache with 16 byte block.Virtual address and context are used in the tag along with protection check.
- Data cache uses write allocate scheme for store miss handling.
- A 4-entry write buffer with 64 bit data field is incorporated.
- Single cycle store for Dcache
- No flash clear
- Dcache streaming is implemented.

1.2.6 MMU:

- 64 entry fully associative TLB.
- Last access of the physical address for instruction cache, and data cache are buffered.
- 2 page hit registers are provided.
- Support virtual address cache
- Improved DVMA performance via larger burst sizes, and interruptible table walk logic.
- The TLB replacement algorithm will be based on pseudo-random linear counter scheme. Some TLB entries can be locked down.
- A programmable bit will be provided to lock down the first 3 entries of the TLB for PTP.

1.2.7 MEMIF:

- Support 8 RAS lines

- Support refresh rate with clock frequency from 50 Mhz and higher.
- Support self-refresh for power down mode of battery based system
- Support low refresh DRAM
- Support for 70/85/100/125Mhz operation
- 24 bit frame buffer support using the Local Graphics protocol.

1.2.8 SBC:

- Support 32 byte transfer to memory, 64 byte transfer between SBus devices
- Support programmable SBus clock
- Support pipeline memory and SBus operation for 32 byte block aligned DVMA write.
- Support pipeline bus arbitration and memory data transfers. (DVMA write)

1.2.9 Jtag operation

- Implement CLK_RST instruction to reset the clk_cntl block to a known state.

1.3 microSPARC-II Memory Map

The microSPARC-II physical memory address has been remapped to conform to the requirement of the Local Graphics bus. The mapping scheme is as follows:

PA [30:28]	Space
000	Memory(256M)
001	Control(256M)
010	Local Graphics Frame buffer (256M)
011	I/O Space (SBus slave select 0)
100	I/O Space (SBus slave select 1)
101	I/O Space (SBus slave select 2)
110	I/O Space (SBus slave select 3)
111	I/O Space (SBus slave select 4)

1.4 Block diagram

The block diagram below shows the major blocks of microSPARC-II. Integrated within microSPARC-II are a SPARC V8 Integer Unit core, a SPARC Reference Memory Management Unit, a Floating Point Unit, Instruction and Data Caches, DRAM controller, and an SBus Controller. Operating at 70Mhz requires an on chip Phase Lock Loop, generating at 2x the chip frequency. As microSPARC-II will be operating from 50Mhz up, the Clock generator will divide the system clock by 3,or 4 to generate the S_bus clock. Programmability is also provided in the MEMIF design to allow varying number of cycle of access to the Dram.

Memory interface is done via 3 major buses:

The MEMIF block interfaces with the DRAM via a "b_memdata<63:0>" bus. The bus is divided into b_memdata_in<63:0> and b_memdata_out<63:0> inside the chip for input and output, respectively.

Cache_fill bus: a 64 bit unidirectional bus from b_memdata_in bus to I and D_caches. This bus is used for cache fill writes, bypass streaming and noncached load/fetches from DRAM.

Misc_bus (Miscellaneous bus): a 32 bit bidirectional bus, used for ASI Ld/St transfers; SBC<->MEMIF transfer(DVMA); SBC<->IU/FPU transfer(PIO); writebuffer->MEMIF transfers.

Figure 1.0 - microSPARC-II Block Diagram

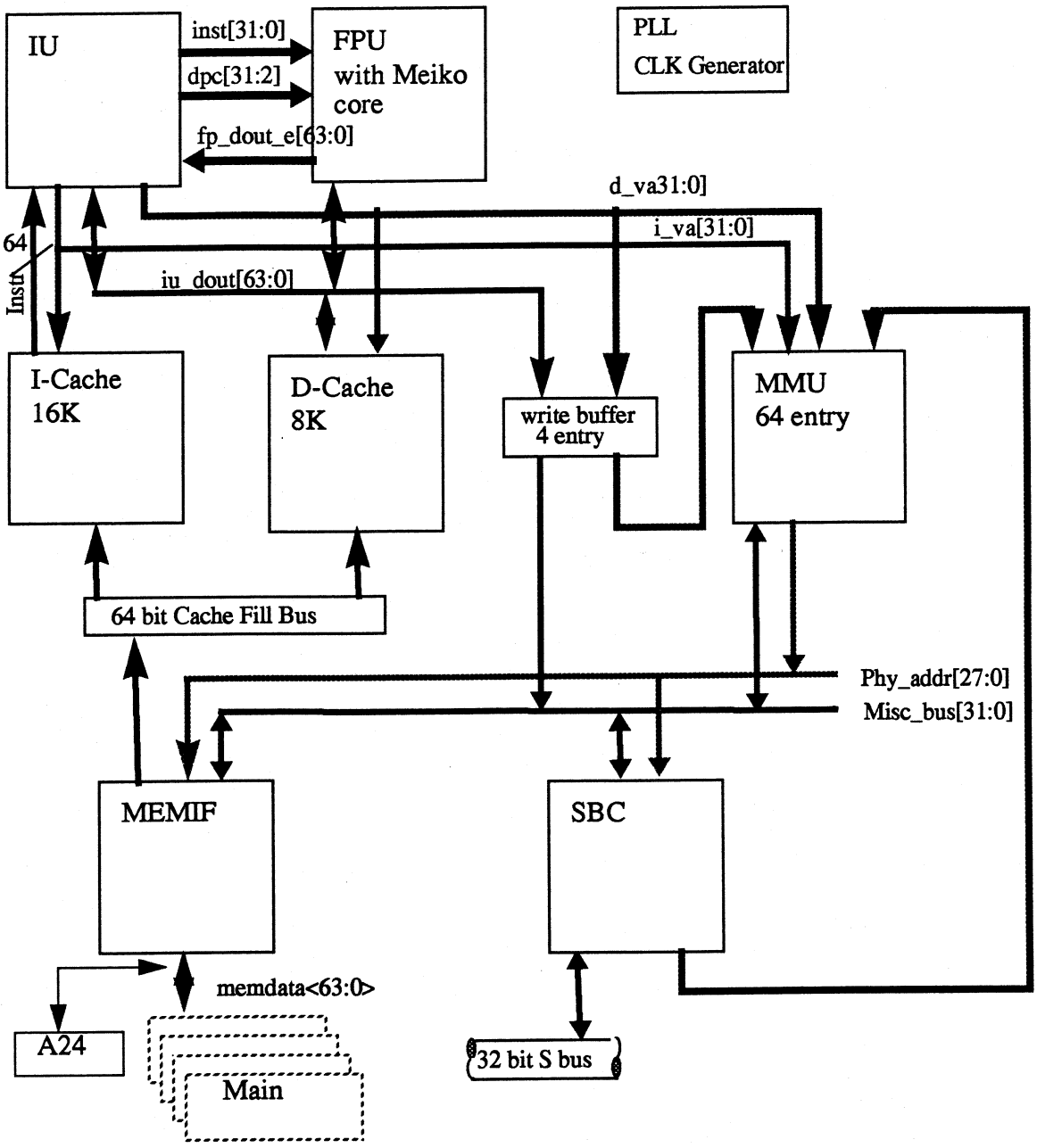
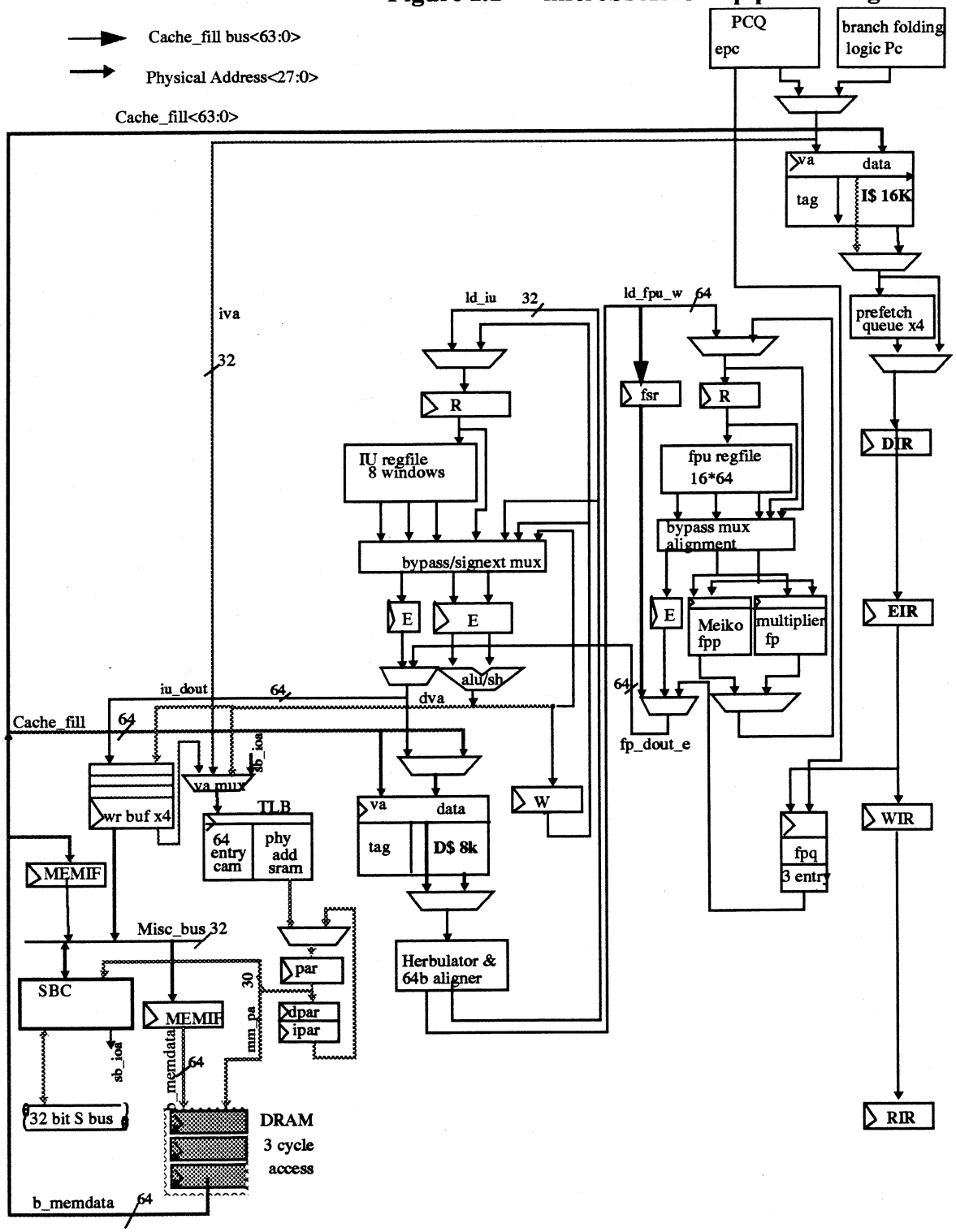


Figure 1.1 - microSPARC-II pipeline Diagram



Chapter 2 CPU Performance

The Performance of the microSPARC-II CPU was characterized using a model, including the caches and the memory subsystem. The 20 programs comprising the SPEC92 benchmark suite were traced and simulated with a microSPARC-II Shadow model (Shadow is an internally-developed instruction trace generator tool). The results were then used to calculate the approximate performance on the SPEC suite. The SPEC suite was compiled with the SPARC compilers 1.0 with the F77 preprocessor from Kuck and Associates. The performance evaluation was done with the following conditions:

- I and D caches are flushed on every system call.
- DRAM page hit will never occur right after system call/DRAM refresh.
- 12 cycle DRAM refresh every 800 cycles.
- 528 cycles of penalty paid on every user window overflow.
- 400 cycles of penalty paid on every user window underflow.
- Cache disturbances due to window overflow and underflow are modeled.
- 40 cycles of penalty are paid on every TLB miss for tablewalk.
- Penalties paid for system calls are based on measured cost (see table). Default cost of system calls not listed is 2000 cycles.
- When the write buffer is full, one entry is flushed. During a cache miss, the write buffer will be emptied. All these flush penalties are included.
- 60ns conventional DRAM's are used.(Fast page mode access time is 30 ns).
- Total trace delay for address from microSPARC-II to DRAM and data from DRAM back to microSPARC-II is 12ns.
- It takes 3 processor cycles to precharge the DRAM and 4 cycles to write to the DRAM from write buffer. (Actual times will be 3.5 cycles each.)
- It takes 4, 5, and 6 processor cycles to read the DRAM from the processor at 70 MHz, 85 MHz, and 100 MHz respectively.

- Instruction queue in IU is not modeled.
- Store byte and store halfword requires reading the memory and modifying it. The memory accesses are accounted for.
- FPU pipeline advances on cache misses, as does the real hardware.
- No guardband included to cover the performance loss due to timing problems encountered later on.
- Numbers marked with '*' are extrapolation from intermediate results, using the curve from a prior simulation of the benchmark.
- Results have been calibrated on a Sunergy system using a microSPARC model derived from the microSPARC-II model. The integer results have been adjusted by -3%, and the FP results by -7%.

Table 1 - Cost Paid per System Call

Call Type	Cycles Paid	Call Type	Cycles Paid
close	17000	sigblock	950
fork	11000	sigsetmask	950
old break	1300	sigstack	7000
open	23000	sigvec	12000
read	15000	write	1500

Table 2 - microSPARC-II Performance with SC1.0 at 70 MHz

Benchmark	Instruction Count (M)	Reference Time (Sec)	CPI	Simulated Spec Ratio
008.espresso	2704	2270	1.30	46.43
013.spice2g6	21294	24000	2.62	29.44
015.doduc	1316	1860	2.73	33.98
022.li	4962	6210	3.01	30.09
023.eqntott	1303	1100	1.28	46.68
026.compress	2621	2770	2.17	33.84
SPECint92				41.39
SPECfp92				40.09

Table 2 - microSPARC-II Performance with SC1.0 at 70 MHz

Benchmark	Instruction Count (M)	Reference Time (Sec)	CPI	Simulated Spec Ratio
034.mdljdp2	3358	7090	2.87	49.93
039.wave5	4407	3700	2.47	22.42
047.tomcatv	1194	2650	3.12	46.35
048.ora	1709	7420	4.36	66.23
052.alvinn	3516	7690	2.24	63.60
056.ear	18058	25500	1.71	54.79
072.sc	3105	4530	1.88	54.95
077.mdljsp2	3170	3350	2.62	27.59
078.swm256	10798	12700	3.01	25.50
085.gcc	4614	5460	2.00	41.45
089.su2cor	4970	12900	4.36	38.95
090.hydro2d	7515	13700	3.27	37.09
093.nasa7	7148	16800	3.12	49.22
094.fpppp	5332	9200	2.50	45.12
SPECint92				41.39
SPECfp92				40.09

Tables 3 and 4 summarize measured microSPARC-II performance at various frequencies. The Spec92 binaries were generated with the Apogee compiler. This compiler gives a 15%-28% performance gain over the SC 1.0 compiler.

Table 3 - microSPARC-II SPECint92 Performance Summary

Frequency [MHz]	Compiler	Estimated SPEC ratio
70	Apogee	54.0
85	Apogee	60.8
100	Apogee	65.9

Table 4 - microSPARC-IISPECfpt92 Performance Summary

Frequency [MHz]	Compiler	Estimated SPEC ratio
70	Apogee	46.1
85	Apogee	53.0
100	Apogee	59.3

2.1 Compiler Optimization Guidelines

This section explains some of the code scheduling issues which affect the performance of the microSPARC-II processor.

2.1.1 Branches

Integer branches can be handled in two ways in microSPARC-II. Branches can be folded with their delay slots or allowed to enter the integer pipeline.

The branch folding is supported by a 4-deep instruction queue. The queue is filled each cycle by a double word fetch. For a branch to be folded, the branch, delay slot, and delay+1 instructions must be in the queue or on the pins of the IU from the instruction cache. In addition,

the instruction preceding the branch cannot be a multi-cycle instruction or a control transfer instruction (CTI), and there cannot be a WRspec (write to a special register) in the pipe.

All branches are predicted taken. The target instruction is fetched the D cycle of the delay slot instruction (or branch-delay slot pair).

```
    bicc    1f
    delay
    delay+1
    ...
1:  target
    ...
```

If the branch can be folded, the branch and delay slot will be executed in cycle x. If the branch is taken, the target will execute in cycle x+1. If the branch is not taken, the target must be killed and delay+1 will be executed in cycle x+2. Thus, folded taken branches take 0 cycles, while folded untaken branches take 1 cycle.

If the branch cannot be folded, it enters into the pipeline in cycle x, and the delay slot in cycle x+1. If the branch was taken, target will execute in cycle x+2. If the branch was not taken, but delay+1 was in the instruction queue, it will execute in cycle x+3, otherwise it must be fetched and will execute in cycle x+4.

	Cycles for Branch	
	Taken	Untaken
Folded	0	1
Not folded	1	1 or 2

2.1.2 Guidelines for Branch Folding

1. Try to make as many bicc's taken as possible. microSPARC-II always predicts taken and fetches the target. If the branch is untaken, it will cost a cycle if it was folded, and may cost a cycle if it wasn't folded.

2. Avoid bicc to bicc control transfers. The target bicc cannot be folded since delay+1 will not be in the instruction queue

```
    bicc    1f
    delay
    ...
1:  bicc    2f
    ...
```

3. Try to have CTI target instructions be double word aligned, e.g.: label 1 is a double word address. This allows the odd word to enter the queue immediately. If the odd word happens to be a bicc, it can be folded. If the target is an odd word, the following bicc will not enter the queue and will not be folded.

```
    bicc    1f
    delay
    ...
1:  target
    bicc    2f
    ...
```

4. Do not put SAVE/RESTORE in delay slot of an annulling bicc. If the SAVE/RESTORE is annulled, microSPARC-II must take a cycle to fix the CWP.

```
    bicc,a  1f
    save
```

5. Do not follow multicycle instructions with a bicc.

Will not fold	Can fold
std	std
bicc	add
delay	bicc
delay	

6. Do not follow WRspec with a bicc. Folding is disallowed when there is a WRspec anywhere in the pipeline's D, E, or W stage. WRspec refers to any of the special registers (psr, wim, tbr, y).

Will not fold	Can fold
mov .., %psr	mov .., %psr
nop	nop
bicc	nop
nop	
nop	
bicc	

Additional notes: Only integer branches are folded. FP branches are not. Calls are not folded due to a RF limitation.

2.1.3 Multicycle Instructions

Most instructions in microSPARC-II take a single cycle to execute. The following instructions take multiple cycles.

Instruction	Cycles
JMP, RETT	2
LDA, STA	2
LDD, LDDA, STD, STDA	2
LDSTB, LDSTBA	2
SWAP, SWAPA	2
STA FLUSH	3
IFLUSH	3
IMUL	19
IDIV	39

2.1.4 Interlocks

microSPARC-II has several pipeline interlocks which may be avoided through code scheduling.

1. Load use interlock - single cycle interlock when an integer load is followed by an instruction which uses the load destination as a source operand.
2. CALL followed by read of r15 - single cycle interlock.
3. RDspec followed by dependent op.
4. Folded SAVE/RESTORE got annulled - cycle to fix CWP (see branch folding).
5. CTI fall through not in queue - see unfolded untaken branch.

2.1.5 Other Guidelines

Use IMUL instruction instead of kernel routine. It is usually better. IDIV is not always better than the kernel divide, so using it is less important.

2.1.6 Floating Point

Scheduling of FP code can have a large impact on FP performance. The most important thing to consider when scheduling FP code is making efficient use of the floating point queue. The FP unit has a three entry floating point queue, and two independent functional units (multiplier and everything else).

microSPARC-II does not interlock for FP loads followed by dependent FPops. This includes double-word loads. FPop operands are read in W-stage, which allows the data to be bypassed from the load to the FP units.

FP Interlocks

1. FP queue full - if an FPop is in E-stage and the FP queue is full, the pipe must be held until the first instruction in the queue completes.
2. FP store waiting for data from FPop in queue - held in E-stage.
3. FP load writing register used by FPop in queue - held in W-stage. This applies whether the FPop register is rs1, rs2, or rd.
4. FPld followed by FPst - single cycle interlock if the FP register (modulo 2) being loaded is the same as the FP register being stored (modulo 2).
5. FPldfsr/FPstdfq followed by any FPop/FPmemop/FPcmp - single cycle interlock.
6. FPop followed by FPldfsr/FPstdfq - LDFSR or STFSR must wait for FPop to complete.
7. FP branch in decode and FCCV (fp condition code valid) deasserted. The IU pipe will interlock until FCCV is reasserted. FCCV is deasserted when an FCMP is started and reasserted when the FCMP completes. The branch is held in D-stage.

Functional Units

There are two functional units: the multiplier and the Meiko core, which handles all other operations. The multiplier can start an operation every three cycles, but operations dependent on the multiplier results must wait five cycles for the result to be written. The initial multiply must also be in the first queue entry if the second multiply is to be started before the first results are written. The Meiko core is not pipelined; when an operation completes, the data and functional unit are both available. See chapter 4 for details on instructional count

FP Queue Details

The FP queue is three entries deep. It allows out-of-order issue, but forces in-order completion. Only one operation can be started per cycle, and only one operation may complete per cycle. An operation does not leave the queue until it has written its results. The following examples demonstrate how dependencies affect the pipeline.

1) Out of order issue, no dependencies - data written back in-order, issued out of order because of functional unit availability

Unit	Issued	Written		
fmuld	%f0, %f2, %f4	Mult	x	x+5
fmuld	%f6, %f8, %f10	Mult	x+3	x+8
faddd	%f12, %f14, %f16	Adder	x+2	x+9

2) FADD throughput - dependencies have no effect. 5 cycles per operation, due to single functional unit.

Unit	Issued	Written		
faddd	%f0, %f2, %f2	Adder	x	x+5
faddd	%f2, %f2, %f0	Adder	x+5	x+10

3) FMUL throughput - no dependency. 3 cycles per operation.

Unit	Issued	Written		
fmuld	%f0, %f2, %f4	Mult	x	x+5
fmuld	%f6, %f8, %f10	Mult	x+3	x+8

4) FMUL throughput - with dependency. 5 cycles per operation.

Unit	Issued	Written		
fmuld	%f0, %f2, %f2	Mult	x	x+5
fmuld	%f0, %f2, %f2	Mult	x+5	x+10

5) FMUL/FADD pair - no dependency. 5 cycles per pair. The second multiply cannot enter the queue until the first add has completed, at which time the second add is being started

Unit	Issued	Written		
faddd	%f0, %f2, %f4	Adder	x	x+5
fmuld	%f6, %f8, %f10	Mult	x+1	x+6
faddd	%f0, %f2, %f4	Adder	x+5	x+10
fmuld	%f6, %f8, %f10	Mult	x+6	x+11

6) FMUL/FADD pair - one dependency. 6 cycles per pair. It doesn't matter which way the dependency goes.

Unit	Issued	Written		
faddd	%f0, %f2, %f4	Adder	x	x+5
fmuld	%f4, %f6, %f8	Mult	x+5	x+10
faddd	%f0, %f2, %f4	Adder	x+6	x+11
fmuld	%f4, %f6, %f8	Mult	x+11	x+16

Unit	Issued	Written		
fmuld	%f0, %f2, %f4	Mult	x	x+5
faddd	%f4, %f6, %f8	Adder	x+5	x+11
fmuld	%f0, %f2, %f4	Mult	x+6	x+11
faddd	%f4, %f6, %f8	Adder	x+11	x+16

7) FMUL/FADD/FMUL - two dependencies. 10 cycles per pair.

Unit	Issued	Written		
faddd	%f0, %f2, %f4	Adder	x	x+5
fmuld	%f4, %f6, %f0	Mult	x+5	x+10
faddd	%f0, %f2, %f4	Adder	x+10	x+15
fmuld	%f4, %f6, %f8	Mult	x+15	x+20

8) Longer instructions (divide, sqrt) - other instructions can enter the pipeline, but none will complete out of order. The integer pipe will not be held unless a fourth FPop tries to enter the queue. Note that the second multiply cannot start until the first advances to the first queue entry.

Unit	Issued	Written		
fdivd	%f0, %f2, %f4	Adder	x	x+35
fmuld	%f6, %f8, %f10	Mult	x+1	x+36
fmuld	%f12, %f14, %f16	Mult	x+36	x+41

2.1.7 Loads and Stores

Load and store ordering was found to have a large impact on microSPARC, the processor from which microSPARC-II is derived. microSPARC-II has a 8 KB write through data cache. If all accesses hit the cache, the order of accesses would make little difference. The order of access can have a large effect on the latency of cache misses. The following guidelines may help improve performance.

1) Group memory accesses by DRAM page. Cache misses require reads from DRAM. The DRAM access is faster if it can be accessed in page mode. Therefore, loads and stores to the same page should be grouped together. One way to do this is to group accesses which use the same base register together, since these are likely to be in the same page. For instance:

Poor order:
```
ld  [%o0], %f3
```

Good order:
```
ld    [%o0], %f3
```

```
ld  [%i5-12], %f4          ld  [%o0+8], %f5
ld  [%o0+8], %f5           ld  [%i5-12], %f4
ld  [%i5-8], %f2           ld  [%i5-8], %f2
```

2) Minimize write buffer full penalty. microSPARC-II has four write buffers. At higher frequencies, the write buffers will take more cycles to flush. So we prefer fewer store instructions if possible, and less clustering of stores to allow the buffers a chance to empty. One technique is to maximize the use of store double (std). A double word store occupies only one write buffer entry and takes one memory access. Storing the two registers separately would require two write buffer entries and two memory accesses.

Since microSPARC-II has four write buffers, up to four stores can be clustered together without stalling the pipe if the stores hit. However, all issued stores must be written to memory before the next cache miss can be processed. It is recommended that the number of instructions between the stores and the next memory access be roughly proportional to the number of stores, to allow time for the write buffer to empty.

3) Minimize use of STB and STH. Memory accesses have word write enables, so these instructions are implemented as a read-modify-write memory operation. This is slower than a normal store.

2.1.8 General Techniques

The following things will help performance, but are not microSPARC-II specific optimizations.

1) Decrease instruction count.

2) Reduce integer load use interlock (better scheduling).

3) Reduce window overflow/underflow..

4) Reduce cache miss rates, especially store miss rates.

2.2 Improved microSPARC-II Performance using the 2 page hit registers

Each time a Virtual Address(VA) is translated, for a memory operation, the resulting Physical Address(PA) is compared to the stored PA of the previous memory operation in the page hit register. If the two PAs are within the same 4K physical address space, the MMU signals that the current operation is a "page hit". This indicates to the memory interface logic that there is no need to toggle the RAS lines to the memory, and the overall access is therefore much faster. Every memory access puts its page address into the page hit register.

In microSPARC-II 2 page hit registers are provided. This allows the saving of two PAs for possible page hits. This also requires the memory interface to divide its memory into two groups, one for each page hit register. Each group has its own RAS lines also. The actual banks of memory are setup so that every other bank/SIMM belong to a given page hit register.

The two page hits registers should especially help applications that alternate a large number of accesses between text and data. In a single page hit register the text and data accesses would thrash the page hit register, reducing its effectiveness.

With the existing page allocation software the first banks pages have to be entirely used or allocated before any of the second banks pages are used. This means that the beneficial effect of having two page hit registers is not seen until that point is reached. This also means that as an application is mapped, the banks will be used serially, one bank after another.

To maximize the benefit of the dual page hit register scheme, the two registers have to be used as much as possible. This means dividing all of the available pages into two groups that correspond to the two page hit registers. After this is done ,a number of allocation preferences can be used. One group can be used for text and the other for data, or simply alternate between the two groups. In microSPARC-II, the DRAM page size is 4K, so alternating could have available a total of 8K of fast access memory at any given time.

The difference between page and non-page access in microSPARC-II is 4 cycles for a page hit versus 11 cycles for non-page hit.

Chapter 3 Integer Unit

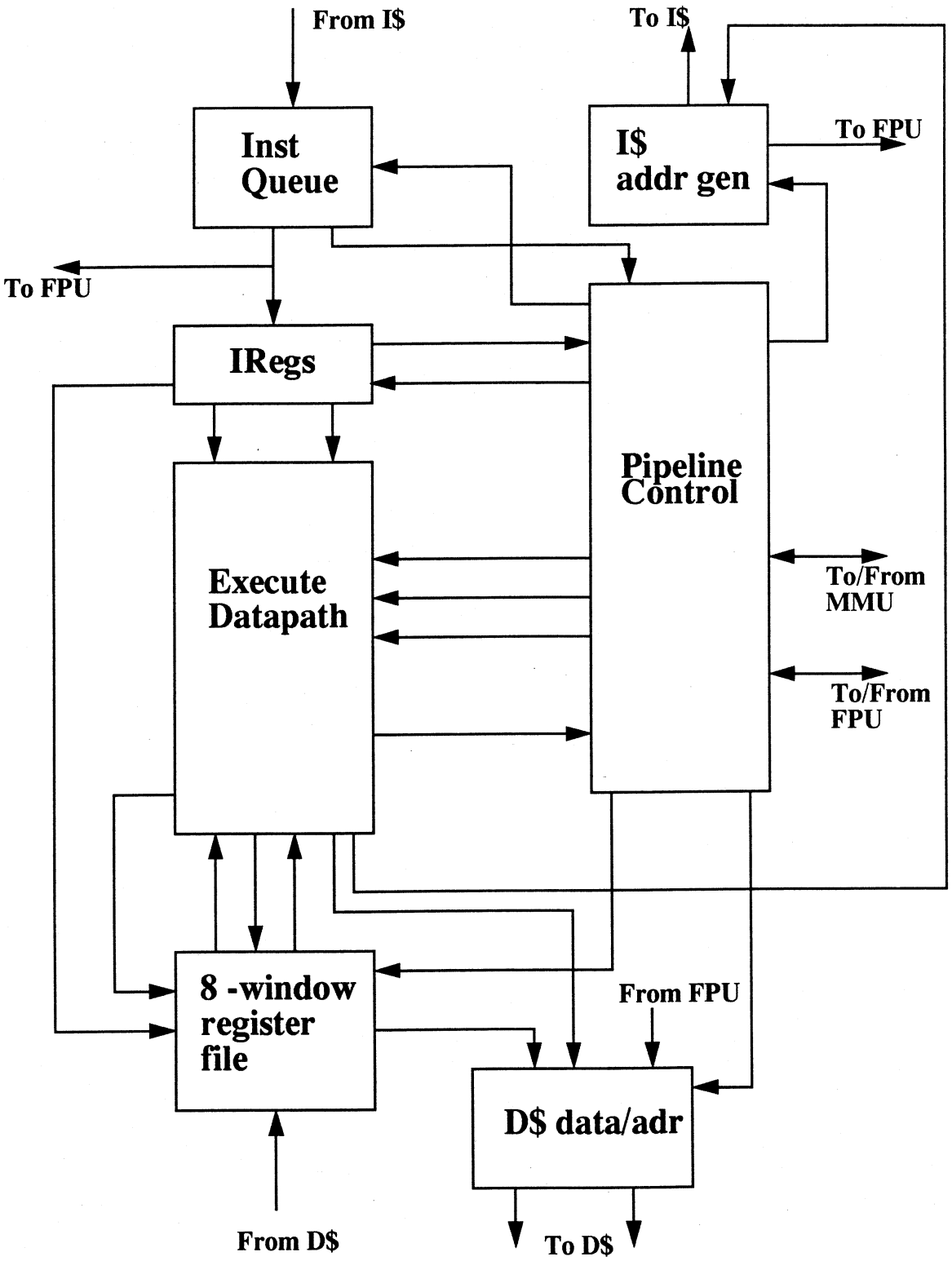
The microSPARC-II Integer Unit is a SPARC integer unit as defined in the SPARC architecture manual (version 8). The IU design goal is to maximize performance, given a constrained die size, using a predefined software architecture. The emphasis is on software compatibility, since the greatest cost impact would be on any software (i.e. kernel, compilers) that would need rewriting.

3.1 Overview

The microSPARC-II Integer Unit is a CMOS implementation of the SPARC 32-bit (Rev 8) RISC architecture. Some important features of this design are:

- 5 stage instruction pipeline
- Branch folding
- Harvard architecture
- Instruction and Data cache streaming support
- IMUL and IDIV implemented in hardware.
- 136-register register file (8 register windows)
- Floating point interface
- 4-deep instruction queue (prefetching)

Figure 3.0 - IU Block Diagram



3.2 Instruction Pipeline

The microSPARC-II IU uses a double (1 branch, 1 other) instruction issue pipeline with 5 stages.

F (Instruction Fetch): Instruction fetch occurs in this cycle. Instructions may be fetched either from the 4-instruction deep queue or directly from the instruction cache. The instruction is valid at the inputs to the IU at the end of this cycle and is registered inside of the IU.

D (Decode): The decode stage is used to decode the instruction and to read the necessary operands. Operands may come from the register file or from internal data bypasses. The register file has 3 independent read ports - two are used for operand or address calculation, and one is used for store operand read in the E-cycle. For situations where the necessary operand is in the pipeline and has not yet been written to the register file, internal bypasses are supplied to prevent pipeline interlocks. In addition, addresses are computed for CALL and Branch in this cycle in the address adder.

E (Execute): The execute stage is used to perform ALU, logical, and shift operations. For memory operations (e.g.: LD) and for JMPL/RETT the address is computed in this cycle. Store operand read is done in this cycle from register file's third read port and sent to the data cache.

W (Write): This stage is used to access the data cache. For cache reads, the data will be valid by the end of this cycle, at which point it is aligned as appropriate. Store data read out in the E-cycle is written to the data cache at this time.

R (Result): This stage loads the result of any ALU, logical, shift, or cache read operation into the register file.

Table 5 - Cycles per Instruction

Instruction	Cycles
Call	1
Single Loads	1
Jump/Rett	2
Double Loads	2
Single Stores	1
Double Stores	2
LDF/LDDF	1
STF/STDF	1
LDA/STA	2
LDDA/STDA	2
STA FLUSH	3
IFLUSH	3
Taken Trap	3
Atomic Load/Store	2
SWAP/LDSTUB	2
IMUL	19/22
IDIV	39/42
All Others	1

3.3 Memory Operations

3.3.1 Loads

All load operations take 1 cycle in the microSPARC-II IU except for LDD which takes 2. For LD, LDB, and LDH the pipeline does the following:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E - Address operands are added to yield the memory address. This address is passed to the cache in this cycle.

W - Address is registered in the cache and access is started. Data is expected at the end of this cycle. Any necessary alignment and sign extension is done by the data cache prior to being registered by the IU.

R - Data is registered in the IU and is written into the register file.

In the event of a cache miss, the miss indication is given to the IU in the W cycle. It is flagged early enough to hold the pipeline. The cache indicates when the miss data is available - the IU can then release the pipeline, register it into the appropriate R cycle register, and write it into the register file. As the cache line is being filled, the IU can accept additional data from either within the filling line or from another line that already exists in the data cache.

An integer LDD takes 2 cycles to complete because of the use of 32 bit datapaths. The pipeline does the following:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E - Address operands are added to yield the even memory address. This address is passed to the data cache in this cycle.

W (E2) - Even memory address is registered in the cache and access is started. This data is sent to the IU. At the same time, the odd address is generated by the IU and sent to the cache.

R (W2) - Even word is registered in the IU and written to the register file. The odd word address is registered in the cache and its access is started.

R2 - Odd word is registered in the IU and written to the register file.

In the event of a cache miss, the miss indication is generated in the W cycle of the LDD. The miss is indicated early enough to hold the pipeline. When the cache sends the correct data, the IU control can release the pipeline, register the even data into the R register, write it to the register file, and prepare to pick up the odd data in the next cycle. As the cache line is being filled, the IU can accept additional data from either within the filling line or from another line that already exists in the data cache.

Floating point load and load double (LDF/LDDF) each operate like an integer load, except that the FPU register file is loaded with the data coming from the data cache. In the case of LDDF, the instruction is still executed in only one cycle due to the 64-bit datapath that exists between data cache and FPU.

3.3.2 Stores

The microSPARC-II IU register file has three independent read ports. As a result, store operations take 1 cycle, except integer STD which takes 2. For ST, STB, STH, and all floating point stores the pipeline does

the following: For integer stores and floating point single stores, the IU will duplicate the store data on both words of the 64-bit bus from IU to data cache. For floating point store double, the words are aligned correctly.

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E - The store virtual address is computed in the ALU. The store operand is read from the third read port of the register file - this includes potential bypassing of results and a store aligner. If it is a floating point store of any size, read out the correct operand(s). Integer and floating point store data are correctly selected and sent to the data cache.

W - The store data is registered by the data cache and written.

R - Store is complete.

For integer STD the pipeline does the following:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E (D2) - The address operands are added to compute the even memory address and this is sent to the data cache. This address will be registered within the IU to provide the data cache with the odd address in the next cycle. At the same time, the even store data is read from the register file's port 3 or bypassed from instructions still in the pipe and is sent to the data cache.

W (E2) - Odd address is sent to the data cache. Odd word is read from register file or bypassed from instructions still in the pipe and is sent to the data cache. Even word is written to data cache.

R (W2) - Odd word is written to the data cache.

R2 - STD complete.

3.3.3 Atomics

SWAP and LDSTUB each take two cycles to complete. The pipeline does the following on the SWAP instruction:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E (D2) - The address operands are added to compute the swap memory address. This address is sent to the data cache to start the cache read portion of the operation. The register to be swapped is read out in this cycle and sent to the data cache.

W (E2) - The data cache returns the memory location accessed. The register to be swapped is sent to the data cache again. (The store address is not sent to the data cache again).

R (W2) - The IU registers the read data and writes it to the register file.

R2 - SWAP complete.

The pipeline does the following on the LDSTUB instruction:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E (D2) - The address operands are added to compute the ldst address. This address is sent to the data cache to start the cache read portion of the operation. 0xffffffff is sent to the data cache along with the appropriate bytemarks for the store.

W (E2) - The data cache returns the memory location accessed and it is shifted appropriately and sent to the IU. 0xffffffff is sent to the data cache again. (The store address is not sent to the data cache again.)

R (W2) - The IU registers the read data and writes it to the register file.

R2 - LDSTUB complete.

3.4 ALU/Shift Operations

Most ALU and shift operations take a single cycle to complete. The exceptions are Integer Multiply and Integer Divide. On Add, Subtract, Boolean, and Shift operations the pipeline does the following:

D - Read operands from register file or bypass from instructions still in the pipe.

E - Do appropriate operation in ALU or shifter. There is a selective inverter on the B input of the ALU to allow for subtracts and certain Boolean operation (e.g. ANDN).

W - Pipe result into R.

R - Write register file with result.

3.5 Integer Multiply

Integer multiply nominally takes 22 cycles to complete, but may complete in 19 cycles if the 3 instructions preceeding the multiply instruction do not write the the integer register file. The algorithm implemented in the microSPARC-II IU is a modified Booth's (2-bit) multiply. The multiply process can be broken up into 4 distinct steps:

Initialization:	1 - 4 cycles
Booth's iteration:	16 cycles
Correction (ala Booth):	1 cycle
Writeback:	1 cycle

The first cycle is used to set up the registers used in the multiply. The rs1 and rs2 registers initialize to the operands of the multiply. The W stage result register and the rs2 register are used as accumulators. At the completion of the multiply, the W register contains the most significant 32 bits of the result and the rs2 register contains the least significant 32 bits of the result. The W register contents are then written to the Y register and the rs2 contents to the destination register in the register file.

3.6 Integer Divide

Integer divide nominally takes 42 cycles to complete, but may complete in 39 cycles if the 3 instructions preceeding the divide instruction do not write the integer register file. If an overflow is detected, the instruction completes in 6 cycles. The algorithm implemented in the microSPARC-II IU is non-restoring binary division (add and shift). The divide process can be broken into 5 distinct steps:

Divide by zero detection:	1 - 4 cycles
Initialization/Ovf detection:	3 cycles
Non-restoring division iteration:	33 cycles
Correction (for non-restoring):	1 cycle
Writeback:	1 cycle

Because the microSPARC-II IU does not allow traps to be taken in the middle of instructions, the first step is to determine if we have a divide by 0 condition.

The high order bits of the dividend are in the Y register. The low order bits are in the rs1 operand. The divisor is in the rs2 operand. In the initialization step, the Y register is read out and put into the rs1 register in the datapath. The rs1 operand is passed through to the W register. The rs2 operand is passed to the rs2 register (surprise!). The W and rs1 registers are used as accumulators. At the completion of the divide, the W register contains the final quotient.

There are two overflow options for signed divide with a negative result defined in the SPARC Rev 8 manual. The microSPARC-II IU generates overflow when:

$$\text{result} < -2^{31} \text{ with remainder} = 0.$$

If an overflow condition is detected, the divide terminates early with the appropriate result being written to the destination register.

If no overflow is detected, the non-restoring (sub and shift) divide stage is started. A correction step is provided to correct the quotient (necessary for this algorithm). After the correction step, the quotient is written to the correct destination register.

3.7 CTI's

3.7.1 Branches

Branches are handled in two ways in microSPARC-II. A branch may be folded with its delay slot or it may flow down the integer pipeline.

In order for a branch to be folded with its delay slot, several criteria must be met. Among these are: the branch, delay slot instruction, and the instruction following the delay slot must all be in the instruction queue or at the inputs of the IU from the instruction cache; no other CTI instruction may be in the D cycle; no multicycle instruction may preceed the branch.

A target instruction fetch is immediately started in the D cycle of the bicc/delay-slot pair. In addition, the delay slot + 1 instruction is sent to a special alternate buffer. All folded branches are predicted taken. In the next cycle, the target instruction may begin operation (if the delay slot is not a multicycle instruction). In this cycle it may be determined that the branch was untaken - this will result in the target instruction being ignored and the delay slot +1 instruction being fetched from the alternate buffer. Taken folded branches require 0 cycles to execute. Untaken folded branches require 1 cycle to execute.

Nonfolded branches usually take a single cycle to execute. There is no penalty for taken vs. untaken branches, even in the event that the instruction prior to the branch sets the condition codes provided the delay slot + 1 instruction is in the instruction queue. In the event that the branch is untaken and the delay slot + 1 is not in the instruction queue, the branch will take two cycles.

In the Decode stage, the IU evaluates the condition codes and branch condition to determine whether it is taken or untaken. The IU outputs the correct instruction address for either the target or fall through paths in time to be registered by the instruction cache for the fetch occurring in the next cycle.

3.7.2 JMPL

JMPL is a two cycle instruction in the microSPARC-II IU.

D - Read operands from register file or bypass from instructions still in the pipe. Sign extend immediate operands. The delay slot instruction is fetched in this cycle.

E (D2) - Compute target address and send this to the instruction cache.

W(E2) - Fetch target.

R (W2) - Load the PC of the JMPL instruction into the destination register.

3.7.3 RETT

RETT is a two cycle instruction in the microSPARC-II IU.

D - Read operands from register file or bypass from instruction still in the pipe. Sign extend immediate operands. The delay slot instruction is fetched in this cycle.

E(D2) - Compute target address and send this to the instruction cache.

W(E2) - Fetch target.

R(W2) - Set PSR.ET to 1, move PSR.PS to PSR.S, and PSR.CWP++.

3.7.4 CALL

CALL is a single cycle instruction in the microSPARC-II IU.

D - Add PC and disp30 to form target address. Send this address to instruction cache. The delay slot instruction is fetched this cycle.

E - The CALL target is fetched.

W - No action.

R - The PC of the CALL is written to r[15].

3.8 Instruction Cache Interface

In the event of an instruction cache miss, the IU is informed of the miss early enough in the F-cycle to prevent the pipeline from moving the missed instruction into D. The IU will wait for the instruction to be fetched and become valid on the instruction bus before allowing the pipeline to continue.

The instruction cache is implemented so that the missed word of the cache line is returned first. The IU is free to stream instructions from the instruction cache as the cache is doing its line fill. This means that the IU is not held for the entire duration of the cache fill, but can use the instructions as soon as either the instruction cache receives it or, if fetching out of the filling line and that line is valid, directly out of the instruction cache. To do this, the IU is told when the instruction addressed by the IU is available to be strobed in. The IU can then selectively hold or release the pipe.

If one of the instructions encountered during the instruction streaming is a taken CTI whose target is outside of the cache line being filled, and if that cache line is valid in the instruction cache, the fetch may take place. If the line is not in the cache, the IU will hold and wait for that line to be filled after the previous line filling is completed.

3.9 Data Cache Interface

The data cache interface is roughly similar to the instruction cache interface. In the event of a data cache miss, the IU will hold the pipeline in the W cycle.

The data cache is also implemented to return the missed word first. On Load instructions, when the data cache indicates that the load data is available, the data is passed through the load aligner (for any necessary alignment) and then the pipe is released, strobing data into the R cycle (and appropriate E cycle) register prior to being written to the register file.

Like the instruction cache, the data cache can return data words as they are being filled. In addition, if, during a fill, a word is addressed from a different cache line, and if the line is valid in the Dcache, that word will be sent to the IU.

3.10 Interlocks

3.10.1 Load Interlock

There is a single cycle load usage interlock in the microSPARC-II IU when a load instruction is followed by an instruction that uses the load operand (data) as a source operand.

3.10.2 Floating Point Interlocks

The IU will interlock the integer pipeline if it detects certain conditions in combinations of FP instructions. These are single cycle interlocks:

Floating point load or load double in the E stage of the pipe, a floating point store or store double in the D stage of the pipe and the FP register number (modulo 2) to be loaded is the same as the FP register number (modulo 2) to be stored.

A LDFSR or STDFQ operation in the E stage of the pipe and the D stage has any FP math operation, an FP compare, or any FP memory operation.

In addition, the IU will interlock when the FPU deasserts the FCCV (Floating point Condition Code Valid) signal and the IU has a floating point branch in D. The IU will continue to interlock the pipe until FCCV is reasserted. The FPU will deassert FCCV when it begins an FCMP instruction and reasserts it when the FCMP is complete.

3.10.3 Miscellaneous Interlocks

Due to the datapath design, the microSPARC-II IU is unable to bypass special register read data to the instruction immediately following it in the pipeline. A single cycle interlock occurs in those cases.

A CALL instruction followed by an instruction that reads r15 (destination register for the CALL), will cause a one cycle interlock.

IMUL and IDIV require datapath structures associated with the register file ports. As a result, they cannot use datapath bypass paths. If the three instructions preceeding the IMUL or IDIV write the register file, the IU will interlock until these instructions have completed. The maximum length of this interlock is 3 cycles. The minimum is 0. (examples of instructions that do not write the integer register file are: stores, FPops, integer and floating point branches, IFLUSH, etc. NOP does write the register file, into Register 0.)

There are also interlocks associated with branch folding. These are dependent on queue, cache, and pipeline state.

3.11 Traps and Interrupts

3.11.1 Traps

The microSPARC-II IU implements all SPARC V8 traps except the following optional traps:

data store error

r register access error

unimplemented FLUSH

watchpoint detected

coprocessor exception

Trap priorities are as defined in SPARC Rev 8. If multiple traps occur during one instruction, only the highest priority trap is taken. Lower priority traps are ignored since it is assumed that lower priority traps will persist, recur, or are meaningless due to the presence of the higher priority trap.

In the pipeline, the trap indication always occurs when the trapping instruction reaches the W stage of the pipeline. Note that traps may be detected as early as the D cycle of the instruction. The trap indication is then piped to the W stage of that instruction.

After the assertion of the TRAP signal, instructions following the trapped instruction in the pipeline and any instructions in the instruction queue are flushed out. The Processor Status Register (PSR) is set as follows: Bit ET (Enable Trap) = 0; Bit PS (Previous Status) = S i.e. the state of the S bit at the time of the trap; Bit S = 1 (Supervisor mode); Bits CWP = Value of Current Window Pointer at the time of the trap. Also field tt (trap type) of the TBR (Trap Base Register) is set to the corresponding trap code and the PC and nPC values at the time of the trap are written into r17 and r18. Instruction fetches then transfer operation to the trap vector as defined in the TBR.

The microSPARC-II IU does not allow traps during execution of multicycle instructions. There are no deferred integer traps. The IU will detect and act on deferred floating point traps.

3.11.2 Interrupts

The microSPARC-II IU is interrupted via the Interrupt Request Level bus. The IU depends on external logic to select the highest priority interrupting device and provide the appropriate IRL level. To discard glitches on the IRL lines, the IU must see at least two cycles where the level on the IRL are the same. Only then does it initiate an interrupt request to the processor. This request is pipelined by one cycle. The interrupt will be taken by the instruction currently in the W cycle of the pipeline (or, if that instruction is a help instruction, by the next non-help W cycle) if the IRL level is greater than the current PIL and there are no higher priority traps that take precedence. A "help" instruction is a dummy instruction inserted whenever additional cycles are required to complete execution of certain instructions, like the second cycle on LDD. The help instruction propagates through the pipeline and maintains its integrity and consistency.

Due to the one cycle delay existing between them when the IRL and PIL are compared and when the trap priorities are checked, this could create a problem where back to back PSR writes could cause an interrupt to occur when the existing value in PSR.PIL is greater than the IRL. The microSPARC-II IU prevents this from happening by hardware.

3.11.3 Reset Trap

On reset, the following steps occur:

Traps are disabled (PSR.ET <= 0) and Supervisor mode is entered (PSR.S <= 1)

If power-up reset:

PSR.PS, PSR.CWP, TBR.TT, r[17] and r[18] are undefined.

Else:

PSR.PS, PSR.CWP, TBR.TT, r[17] and r[18] are unchanged.

Execution begins at location PC=0 and nPC=4.

3.11.4 Error Mode

Error mode is entered when a trap occurs and PSR.ET = 0. Entry into error mode causes the following to occur:

PSR.S <= 1; PSR.PS and PSR.CWP remain unchanged; the contents of PC and nPC are stored into r[17] and r[18]; PC and nPC are set to 0 and 4 respectively and the IU_ERROR signal is asserted.

In addition, the TBR.TT may be changed if the trapping instruction is an RETT. The TBR.TT will reflect:

Privileged instruction trap when PSR.S = 0

Underflow trap when a window underflow occured

Misaligned trap when a misaligned target address occured.

The IU will remain in error mode until it is reset.

3.12 Floating Point Interface

The microSPARC-II IU controls the addresses for all instructions and for floating point memory operations. The IU supplies the fetched instruction directly to the FPU from the instruction queue. The IU also informs the FPU if the instruction just loaded into the instruction register is valid.

For floating point loads, the IU starts the cache access and the FPU reads the data. If the FPload causes a data cache miss, the IU will sequence the cache miss. The FPU will pick up the missed data once the IU releases the pipeline. For floating point stores, the IU starts the cache access and

picks up the store data from the FPU. The IU then muxes this data out onto the data cache store bus.

The IU detects FP conflict cases and interlocks the pipeline. In addition, the FPU may assert FHOLD to hold the IU pipeline when it detects an internal conflict. It will deassert FHOLD when the conflict is resolved.

FCC and FCCV are used by the IU to determine taken and untaken cases for floating point branches. If a floating point branch is detected in Decode and FCCV is not asserted, the IU will interlock until FCCV is asserted.

The FPU asserts the FEXC line when it detects a floating point exception. The IU will acknowledge the floating point exception (FXACK) when a floating point instruction is in the W stage of the pipe and the IU takes a floating point exception trap.

FPops take one cycle in the IU, plus additional cycles in the FPU. For the number of cycles in the FPU, please refer to the FPU section in this document.

3.13 Special Features

The microSPARC-II IU has some built in features to make debug and bringup easier.

The IU is fully scanned, with all registers connected into the microSPARC-II scan chain (JTAG).

Certain registers of the scan chain are accessible only through the scan chain. These enable certain features useful for bringup and debug.

RF bypass - each read port has a bypass enable that causes the write data to be bypassed to the read port. Two registers in the scan chain can be set to enable this. These registers will be zeroed immediately on the next clock (when scan mode is off), disabling this feature.

Illegal opcode event - when this feature is enabled through the scan chain, the IU will assert the iu_event signal when a certain illegal opcode is decoded in the pipeline and the instruction causes an illegal instruction trap. The opcode in question is op=10 binary and op3=111111 binary. Once enabled, this feature can only be cleared through the scan chain.

IU error event - when this feature is enabled through the scan chain, the IU will assert the iu_event signal when it enters error mode. Once enabled, this feature can only be cleared through the scan chain.

An additional scan access only register enables branch folding to occur out of the instruction queue. When disabled, branches will not be folded.

3.14 Compliance with SPARC version 8

The microSPARC-II IU has been designed to comply with the SPARC V8 architecture, including hardware integer multiply and divide. MicroSPARC-II does deviate from full support of V8 features in the following ways:

Alternate space memory operations proceed normally with a single caveat. Rather than the 8 bits of ASI, the microSPARC-II MMU only decodes 6 bits and the remaining 2 most significant bits are ignored. Therefore, out of bound ASI encodings are not detected.

The microSPARC-II IU does not implement STBAR since there is no need to force store ordering in this system. STBAR will pass through the pipe as a Read Y Register operation with destination being the bit bucket (%g0).

The microSPARC-II IU does not support reads and writes to the Ancillary State Registers.. All read cases will act like a Read Y Register operation. All write cases will act like a NOP.

When entering Error Mode, the microSPARC-II IU decrements the CWP and updates r17 and r18. While not in conflict with the Rev. 8 specification, it is noted here.

The value read from the implementation field (IMPL) of the PSR for microSPARC-II is (hexadecimal) 0. The value read from the version field of the PSR is (hexadecimal) 4.

Chapter 4 Floating Point Unit

The microSPARC-II floating-point unit (FPU) consists of two main functional blocks: the floating-point controller (FPC) and floating-point processor (FPP).

The FPC controls the dispatch of instructions from the IU to the FPP; controls data transfer between the data cache, floating point register file, and FPP; detects data dependencies; and generates exception handshaking. The FPP performs the floating-point operations. The microSPARC-II FPP contains the Meiko core and a separate fast multiply unit. The Meiko core is a design licensed from Meiko Inc.

4.1 Overview

The microSPARC-II FPU fully executes all single and double precision FP instructions as defined in the SPARC Architecture Version 8. All other FP instructions (including quads) trap to unimplemented. All implemented instructions, except FsMULd, will complete in hardware for all cases. Therefore the microSPARC-II FPU could only generate an unfinished exception for FsMULd.

The Meiko FPP implements the following algorithms, which contribute to the final size and speed of the FPU.

- 8-bit multiply step
- 2-bit division step
- 1-bit square root step
- short distance (0-15 bits) shifter/normalizer
- separate single cycle round step
- microcode state machine to control FPP and decode operation.

The fast multiplier implements FMULs, FMULd and FsMULd. In most cases, this operations can take place while, in parallel, the Meiko core executes other FP instructions (like FADD for example), providing significant throughput when executing popular algorithms.

In some exceptional cases, the multiplier can not complete a multiplication that was started. In those cases, the operation is aborted and restarted in the Meiko unit. In all cases, execution in the right sequence is guaranteed.

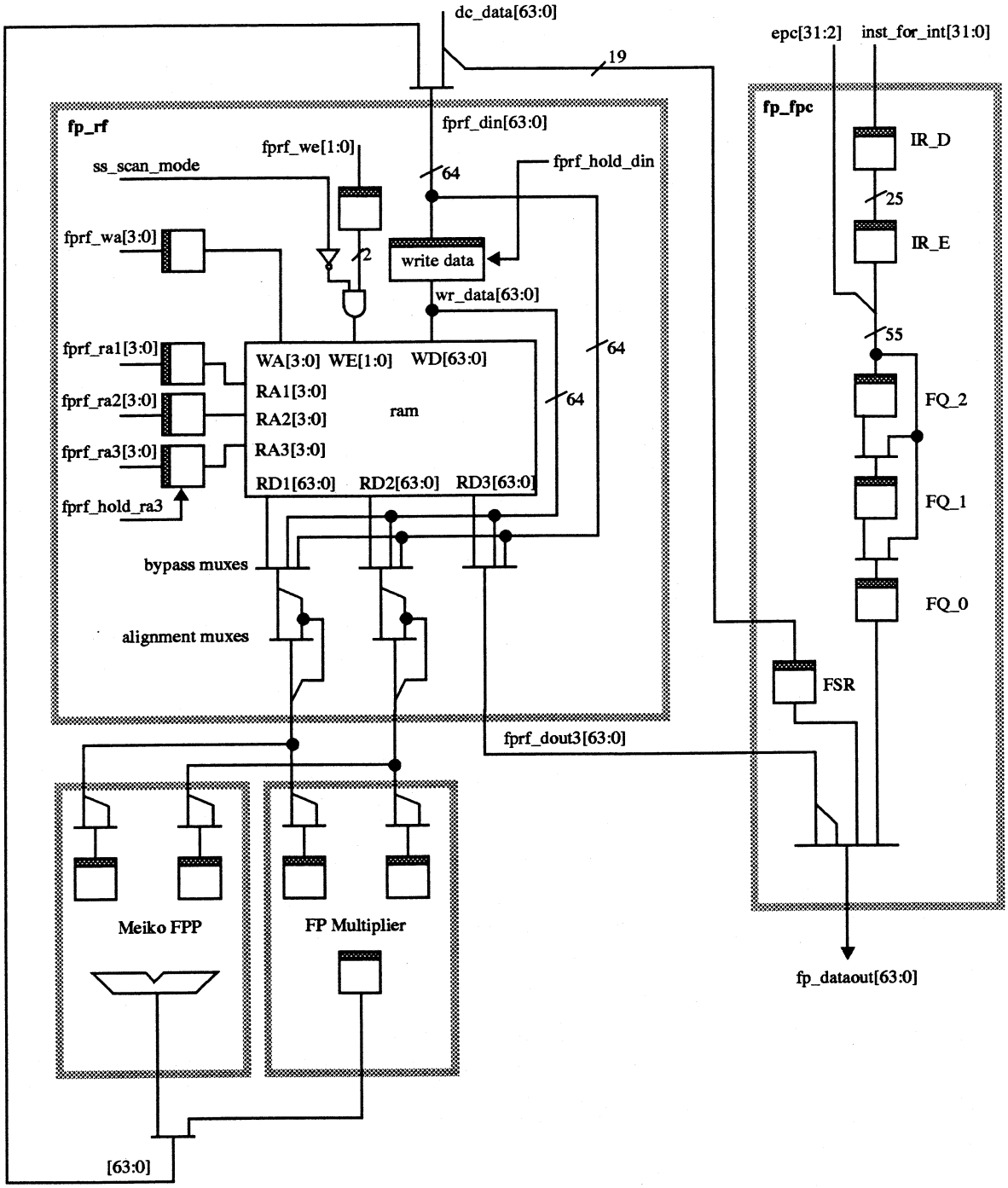


Figure 4.0 - FPU Block Diagram

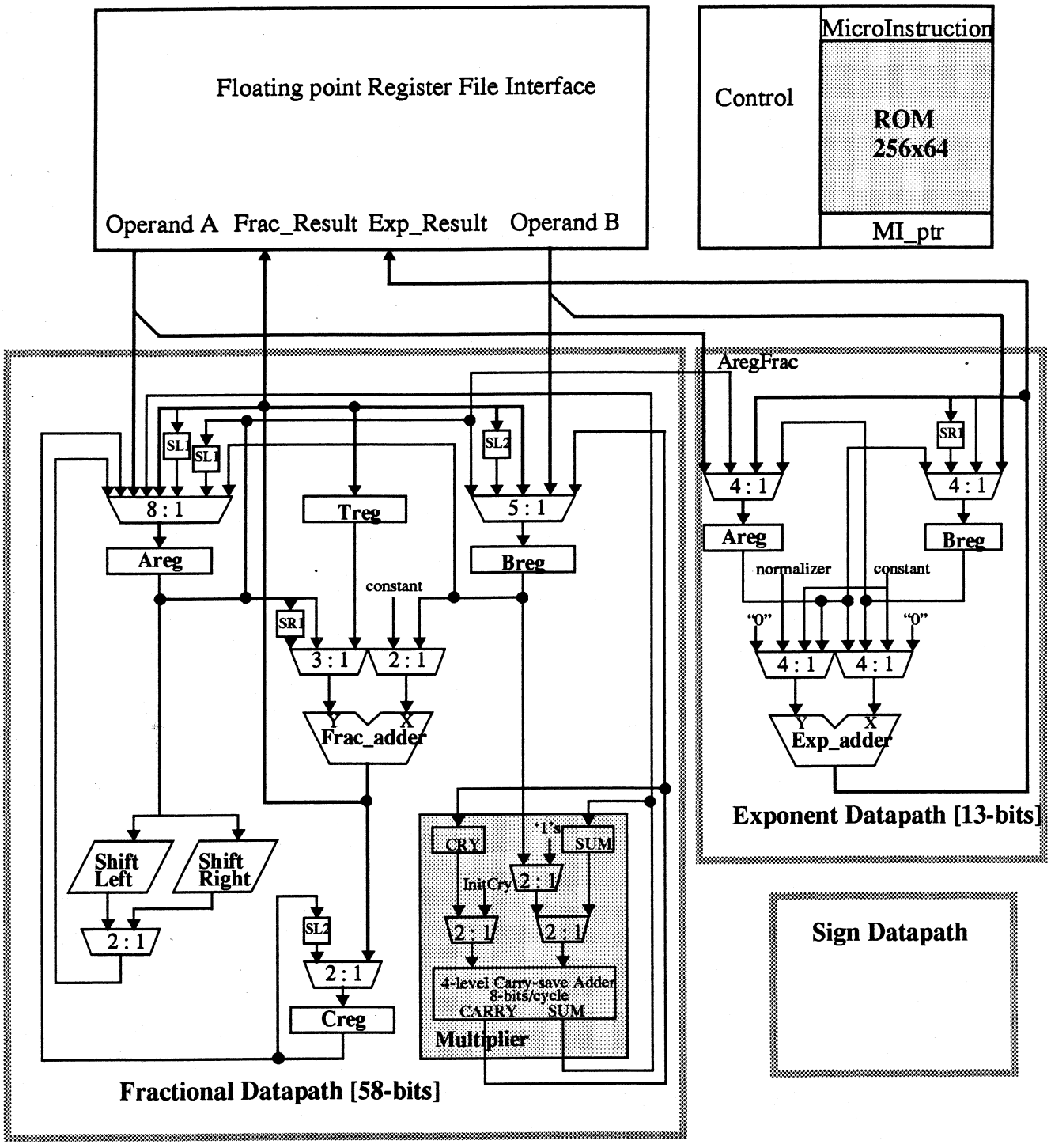


Figure 4.1 - Meiko FPP Block Diagram

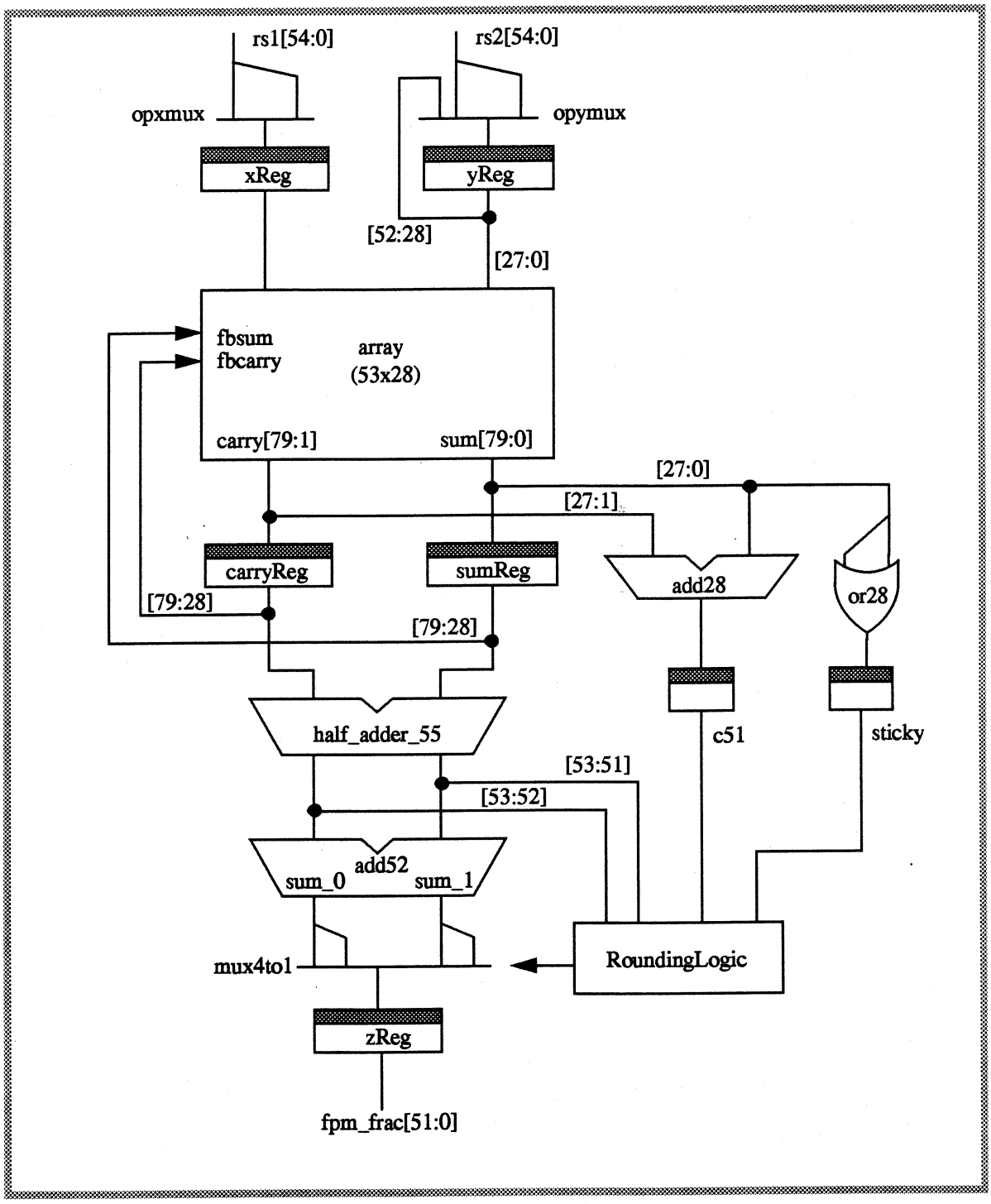


Figure 4.2 - microSPARC-II Multiplier Mantissa Block Diagram

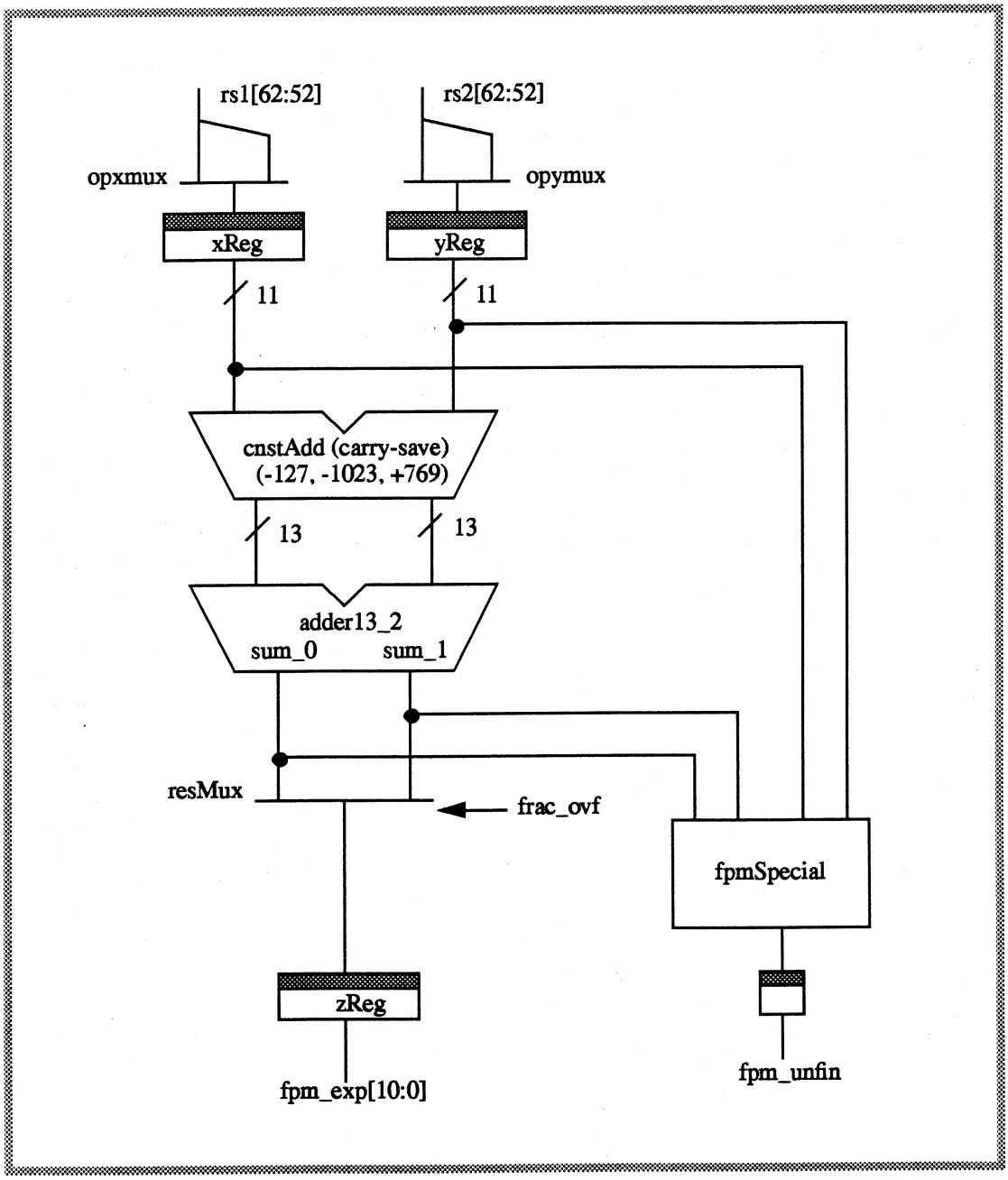


Figure 4.3 - microSPARC-II Multiplier Exponent Block Diagram

4.2 FPU Internal Information

The FPC logic is partitioned into 4 main sections:

1. FHOLD generation
   - FQ full, and FPop in pipeline (held in E-stage)
   - dependent fp store (held in E-stage)
   - dependent fp load (held in W-stage)

2. FQ load control
   - An FPop can be loaded into the FQ if an entry is available and the FPU is in fp_execute mode.

3. FQ issue control
   - An FPop can be issued to the FPP if it is in the FQ or E-stage of the pipeline, and the FPP is not busy, and there are no data dependencies, and the FPU is in fp_execute mode.

4. FQ writeback control
   - An FPop can write back its result if: it is in the front entry of the FQ, the FPP has finished execution and the FPU is in fp_execute mode

The figures below show internal state diagrams and waveforms for some control signals.

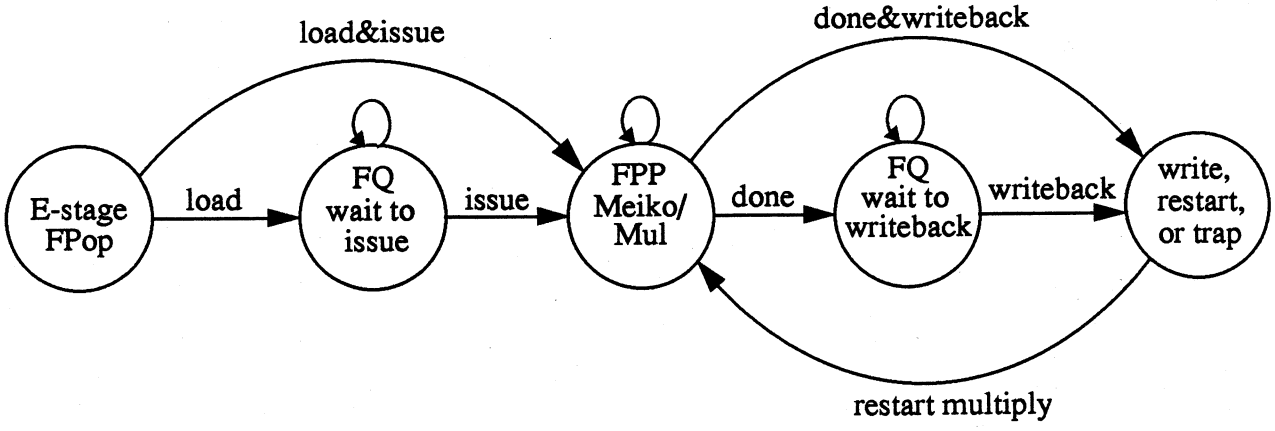


Figure 4.4 - FPU Internal Control Flow Diagram

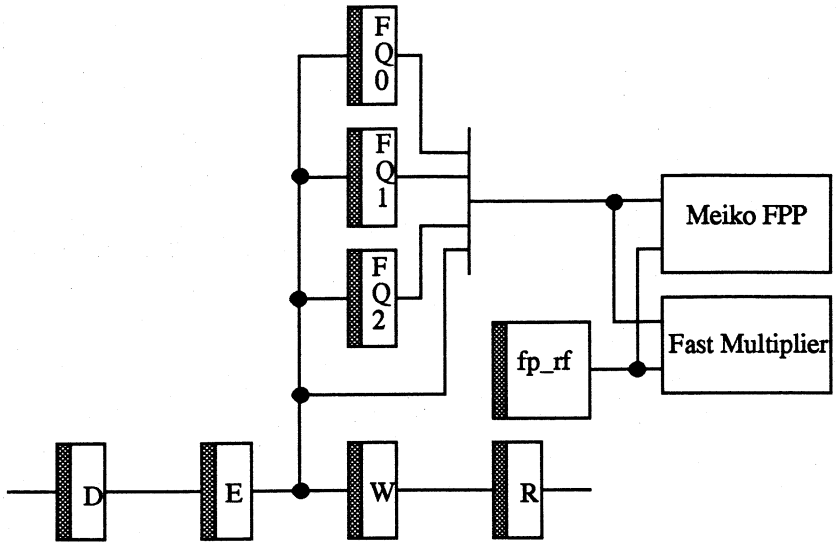


Figure 4.5 - FPU Instruction Pipeline Diagram

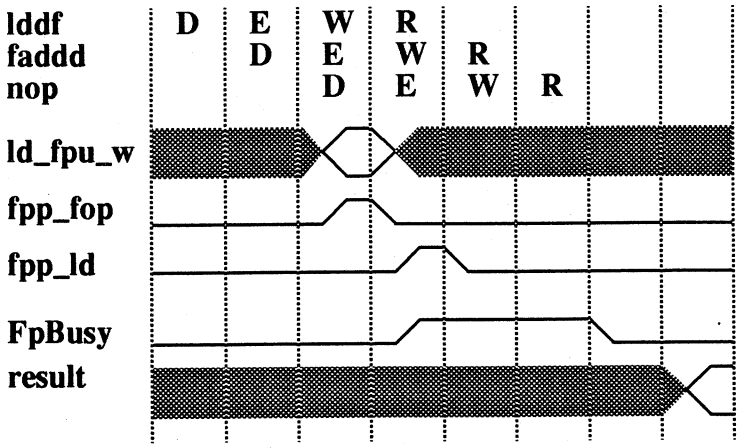


Figure 4.6 - FPC/Meiko FPP Interface Waveforms

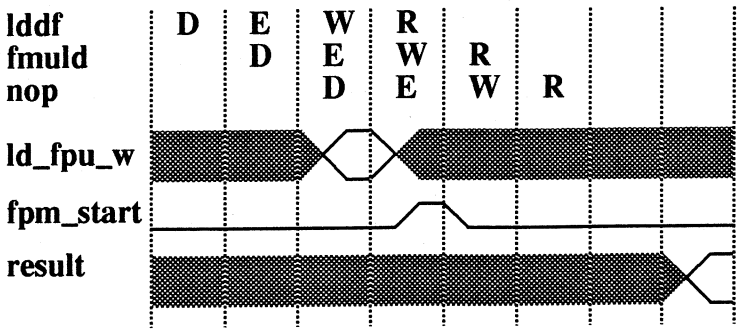


Figure 4.7 - FPC/Multiplier FPP Interface Waveforms

4.3 Deviations from SPARC V8

The microSPARC-II FPU deviates from SPARC V8 by not supporting quad-precision floating-point operations. It traps to unimplemented when these instructions are encountered.

The microSPARC-II FPU also differs from the "SPARC IEEE 754 Implementation Recommendations" (Appendix N in SPARC Architecture Manual V8) in the NaN format. The following figure shows the value returned for an untrapped floating-point result in the same format as the operands:

		rs2 operand		
		number	QNaN2	SNaN2
rs1 operand	none	IEEE 754	QNaN2	ME_NaN
	number	IEEE 754	QNaN2	ME_NaN
	QNaN1	QNaN1	QNaN1	ME_NaN
	SNaN1	ME_NaN	ME_NaN	ME_NaN

Figure 4.8 - Untrapped FP Result in Same Format as Operands

In the figure above, all QNaN results will have their sign bit set to 0. ME_NaN is 0x7fff0000 (single-precision) or 0x7fffe00000000000 (double-precision).

	operand (rs2)			
operation	+QNaN	-QNaN	+SNaN	-SNaN
fstoi	ME_NaN	-imax	+imax	-imax
fstod	(QNaN2)	(QNaN2)	ME_NaN	ME_NaN
fdtos	ME_NaN	ME_NaN	ME_NaN	ME_NaN
fdtoi	ME_NaN	-imax	+imax	-imax

Figure 4.9 - Untrapped FP Result in Different Format

In the figure above, +imax = 0x7fffffff, and -imax = 0x80000000. (QNaN2) is a copy of the mantissa bits of the operand, with the extra low order bits zeroed, and the sign bit zeroed.

4.4 Implementation Specific Features

The microSPARC-II FPU implements a 3-entry floating-point deferred trap queue. When a floating-point instruction generates an fp_exception, microSPARC-II will delay the taking of the fp_exception trap until the next floating-point instruction is encountered in the instruction stream. The microSPARC-II FPU implementation can be modeled as having 3 states: fp_execute, fp_exception_pending, and fp_exception. These are shown in the figure below.

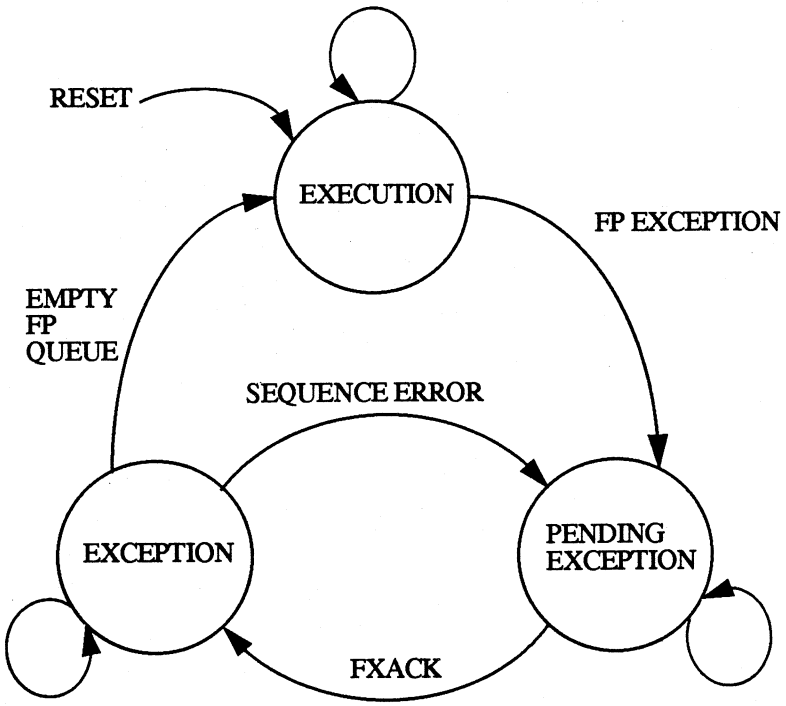


Figure 4.10 - FPU Operation Modes

Normally the FPU is in fp_execute state. It moves from fp_execute to fp_exception_pending when an FPop generates a floating-point exception.

The FPU moves from fp_exception_pending to fp_exception, when the IU attempts to execute any floating-point instruction (including fbcc's). This transition (FXACK) generates an fp_exception trap. At this time the first FQ entry contains the instruction and address of the FPop which originally caused the fp_exception.

An fp_exception trap can only be caused while the FPU is moving from the fp_exception_pending state to the fp_exception state (or by executing a STDFQ instruction when FSR.qne == 0, as described below). While in fp_exception state, only floating-point store instructions may be executed (particularly STDFQ and STFSR) and they can not cause an fp_exception trap.

The FPU remains in the fp_exception state until the FQ is emptied by executing STDFQ instructions. At that time, the FPU returns to the fp_execute state.

If an FPop, or a floating-point load instruction (excluding fbcc's and all store instructions) is executed while the FPU is in fp_exception state, the FPU returns to fp_exception_pending state and also sets the FSR.ftt field to sequence_error (0x4). The instruction that caused the sequence_error is not entered into the FQ.

If a STDFQ instruction is executed when the FQ is empty (FSR.qne == 0, FPU is in fp_execute state), the FPU will generate an immediate fp_exception trap (not deferred) and set the FSR.ftt field to sequence_error (0x4), but the FPU will remain in fp_execute state.

The STDFQ instruction will store the address from the FQ to the effective address, and the instruction from the FQ to the effective address + 4.

4.5 Software Considerations

This section describes the software visible features of the microSPARC-II FPU.

The FSR.ftt field is set whenever an FPop completes or causes an exception. This field will remain unchanged until another FPop completes (or causes a sequence error). The FSR.ftt field may be cleared by executing a non-trapping FPop, such as fmovs%f0,%f0.

The following table describes the bits in the Floating-Point Status Register (FSR):

FSR Bits	field	values	Description	writeable by LDFSR
31:30	RD	0 - Round to nearest (tie-even) 1 - Round to zero 2 - Round to +infinity 3 - Round to -infinity	Rounding Direction	Yes
29:28	res	always 0	reserved	No
27:23	TEM	0 - disables corresponding trap 1 - enables corresponding trap	Trap Enable Mask	Yes
22	NS	always 0	Nonstandard FP	No
21:20	res	always 0	reserved	No
19:17	ver	always 4	FPU version number	No
16:14	FTT	0 - None 1 - IEEE Exception 2 - Unfinished FPop 3 - Unimplemented FPop 4 - sequence error	FP trap type	No
13	QNE	0 - queue empty 1 - queue not empty	Queue Not Empty	No
12	res	always 0	reserved	No
11:10	FCC	0 - == 1 - < 2 - > 3 - ? (unordered)	FP Condition Codes	Yes
9:5	AEXC	0 - no corresponding exception 1 - corresponding exception	Accrued Exception Bits	Yes
4:0	CEXC	0 - no corresponding exception 1 - corresponding exception	Current Exception Bits	Yes

Table 6 - FSR Summary

4.6 FP Performance Factors

The microSPARC-II FPU instruction cycle counts are provided in the following table. The counts are in CPU cycles.

Instruction	Min	Typ	Max
fadds	4	5	17
faddd	4	5	17
fsubs	4	5	17
fsubd	4	5	17
fmuls	3	3	28
fmuld	3	3	35
fsmuld	3	3	3
fdivs	6	20	38
fdivd	6	35	56
fsqrts	6	37	51
fsqrtd	6	65	80
fnegs	2	2	2
fmovs	2	2	2
fabss	2	2	2
fstod	2	2	14
fdtos	3	3	16
fitos	5	6	13
fitod	4	6	13
fstoi	6	6	13
fdtoi	7	7	14
fcmps	4	5	15
fcmpd	4	5	15
fcmpes	4	5	15
fcmped	4	5	15
unimplemented	3	3	3

Table 7 - FPU Instruction Cycle Counts

Because of the limited shifter size (0-15 bits was chosen to save hardware), the fpu instruction cycle counts are data dependent. There are 5 ways in which operations **may** take longer than the typical cycle count:

1. Exceptional operands (such as NaN, etc.) may add several cycles to the typical cycle count. In a normal environment, these are rare events probably caused by ill-conditioned data and will be trapped (if traps are enabled).

2. Possible exceptional results (results which are very close to underflow or overflow) may add up to 5 cycles to the typical cycle count. In a normal environment these are rare events, probably caused by ill-conditioned data.

3. Denormalized operands will add 1 extra cycle for each 15 bit shift required to normalize before the operation, and 1 extra cycle for each 15 bit shift required to denormalize the result after the operation (if necessary). Because operations on denormalized numbers will always complete in hardware (except for the FsMULd instruction), the overall performance will be greater than for an fpu which traps on denormalized operands.

4. Add or Subtract which require an initial alignment of more than 15 bits will add 1 extra cycle for each 15 bit shift. Also, a Subtract result which requires a shift of more than 15 bits to normalize will add 1 extra cycle for each 15 bit shift.

5. Non-standard rounding modes (RZ and RN are the typical operating modes) may require up to 3 additional cycles for some corner cases and exceptions.

Statistical analysis shows that, on average, 90% of fpu instructions will complete with the typical cycle count.

For a more detailed description of the Meiko FPP, please refer to the Meiko FPU specification, provided by Meiko Limited of Bristol, England.

The figures below show the peak performance (cached) of the microSPARC-II FPU for certain interesting FPop combinations.

```
faddd  %f0, %f2, %f4
faddd  %f6, %f8, %f10
faddd  %f12, %f14, %f16
  .                              5 cycles each =
  .                              14.0 MFLOPS @ 70MHz
  .
faddd  %f0, %f2, %f4
```

Figure 4.11 - FP add peak performance

```
fmuld  %f0, %f2, %f4
fmuld  %f6, %f8, %f10
fmuld  %f12, %f14, %f16
  .                              3 cycles each =
  .                              23.3 MFLOPS @ 70MHz
  .
fmuld  %f0, %f2, %f4
```

Figure 4.12 - FP mul peak performance (no dependencies)

```
fmuld  %f0, %f2, %f2
fmuld  %f0, %f2, %f2
fmuld  %f0, %f2, %f2
  .                              5 cycles each =
  .                              14.0 MFLOPS @ 70MHz
  .
fmuld  %f0, %f2, %f2
```

Figure 4.13 - FP mul peak performance (dependency)

```
fmuld  %f0, %f30, %f0
faddd  %f10, %f12, %f12
fmuld  %f2, %f30, %f2
  .                              5 cycles per pair =
  .                              28.0 MFLOPS @ 70MHz
  .
fmuld  %f0, %f30, %f0
```

Figure 4.14 - FP mul-add peak performance (no dependencies)

```
fmuld  %f0, %f30, %f0
faddd  %f0, %f12, %f12
fmuld  %f2, %f30, %f2
  .                              6 cycles per pair =
  .                              23.3 MFLOPS @ 70MHz
  .
fmuld  %f0, %f30, %f0
```

Figure 4.15 - FP mul-add peak performance (dependency)

Chapter 5 Memory Management Unit

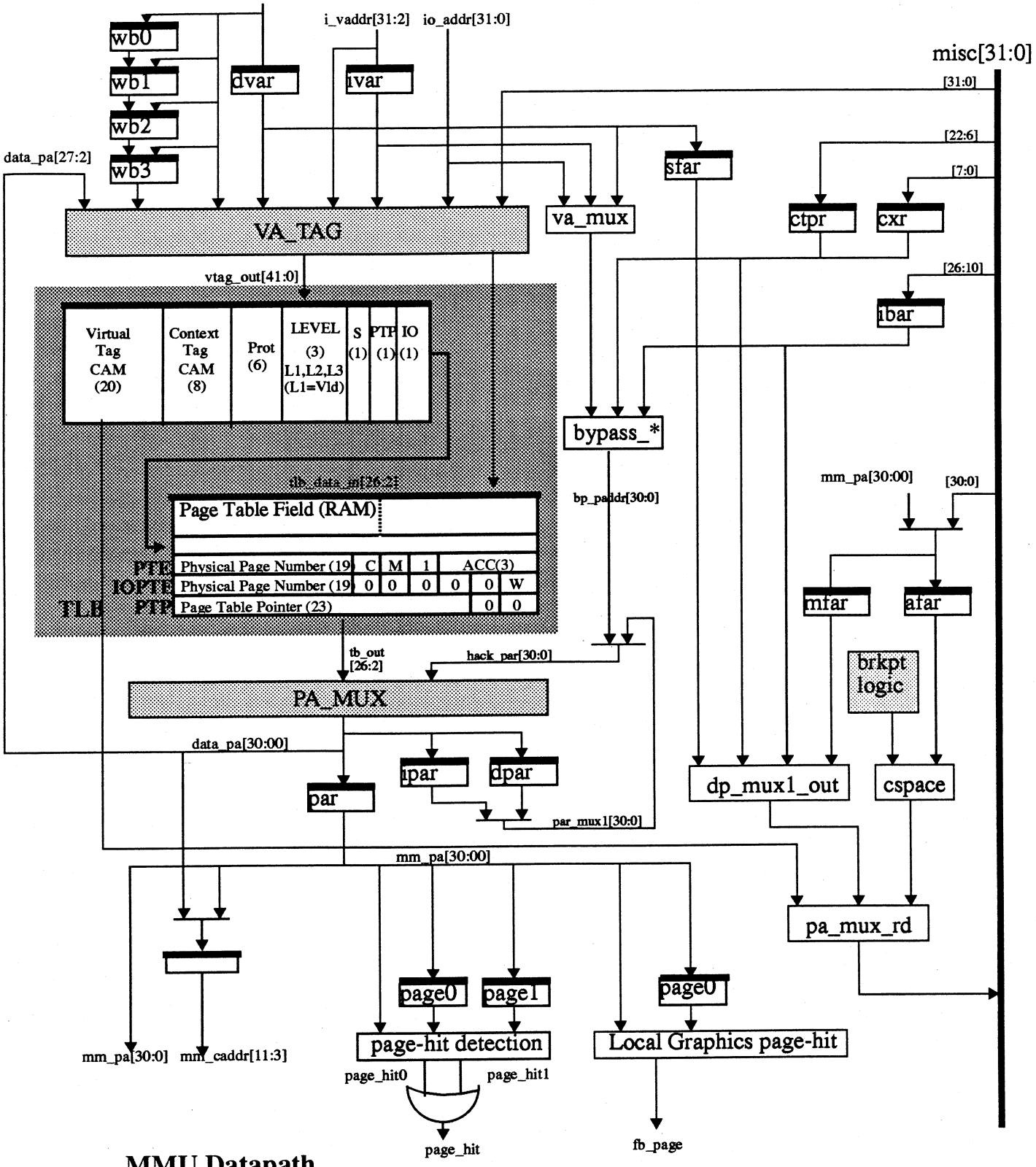
The microSPARC-II MMU provides the functionality of both a reference MMU as specified by the SPARC Reference MMU Architecture and an IOMMU. Additionally, much of the memory arbitration logic is contained within the MMU block.

5.1 Overview

The MMU provides four primary functions. First, the MMU translates virtual addresses of each running process to physical addresses in memory. More specifically, the MMU provides translation from a 32 bit virtual address to a 31 bit physical address by using a translation lookaside buffer (TLB). The 3 high order bits of Physical Address are maintained to support memory mapping into 8 different address spaces. The MMU supports the use of 256 contexts. Second, the MMU provides memory protection so that a process can be prohibited from reading or writing the address space of another process. Page protection and usage information is fully supported. Third, the MMU implements virtual memory. The page tables are maintained in main memory. When a miss occurs in the TLB the table walk is handled in hardware and a new virtual to physical address translation is loaded into the TLB. Finally, the MMU performs the arbitration function between IO, Data Cache, Instruction Cache, and TLB references to memory.

The microSPARC-II MMU contains a 64 entry fully associative TLB and uses a pseudo random algorithm for the replacement of TLB entries. An address and data path block diagram follows.

Figure 5.0 - MMU Address and Data Path Block Diagram



MMU Datapath

5.2 Translation Lookaside Buffer

The TLB is a 64 entry, fully associative cache of page descriptors. It caches virtual to physical address translations and the associated page protection and usage information. The pseudo random replacement algorithm determines which of the 64 entries should be replaced when needed. In the descriptions that follow the terms VA and PA are used to generically describe any virtual address (sb_ioa, wb_vaddr, i_vaddr or d_vaddr) or physical address (mm_pa, or mm_caddr) respectively.

5.2.1 TLB Replacement

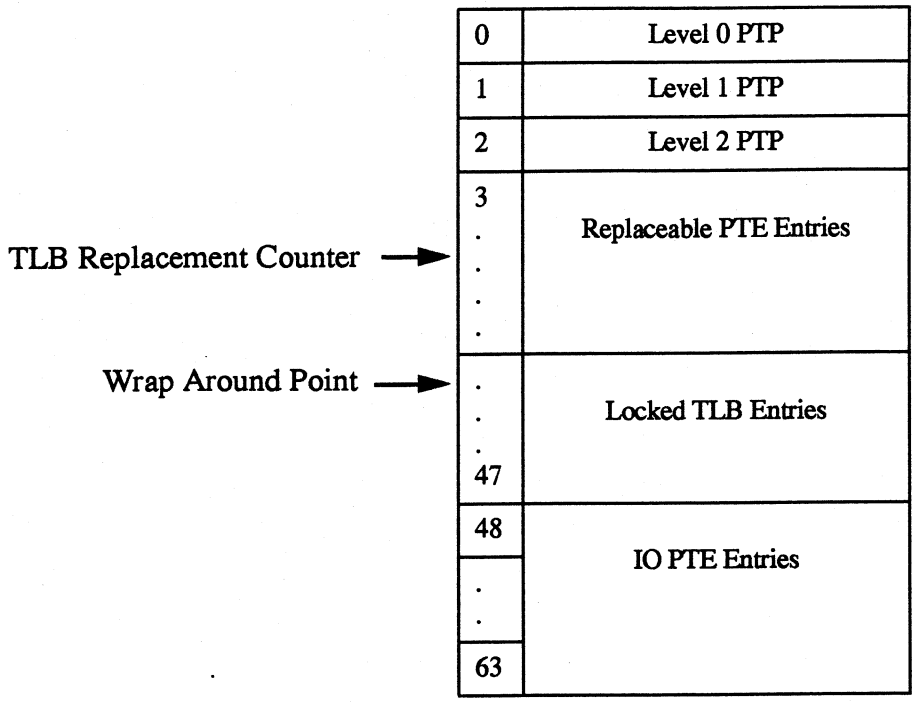
The TLB uses a pseudo random replacement scheme. There is a 6 bit counter in the TLB Replacement Control Register (TRCR) which is incremented by one during each CPU clock cycle to address one of the TLB entries. When a TLB miss occurs, the counter value is used to address the TLB entry to be replaced. On reset the counter is initialized to zero. There is also a bit in the TRCR which is used to disable the counting function. A simple diagram follows. Additionally the TRCR has a programmable 6-bit field which defines the counter "roll-over" point. This effectively locks down entries beyond the roll-over point, and prevents their replacement by subsequent translations.

In addition to this feature the MMU TLB also support the automatic partitioning of entries between IO PTEs and memory PTEs. When enabled via the TLB Replacement Control Register, the TLB will place all IO PTEs in entries 48-63. Thus preventing the displacement of application PTEs by I/O activity, and vice versa.

The last "entry locking" mechanism that is supported by the MMU TLB is the placement of PTPs. Again when enabled via the TLB Replacement Control Register, the MMU will reserve three entries for the exclusive use of PTPs. PTPs would then be limited to those three entries, preventing PTPs from displacing any existing PTEs. This also prevents the displacement of PTPs by PTE entries.

The MMU has the ability to store level 2 PTPs with a virtual tag or a physical tag. This is controlled via a bit in the TLB Replacement Control Register. When physical tags are enabled the MMU table walk algorithm will start with a root level access if a PTE was not found on the initial look-up. If, however, virtual tags are enabled for PTP2, the table walk algorithm will search for a virtually tagged PTP2 following the initial PTE miss. Should the virtually tagged PTP not be found the root level walk is started. When a virtually tagged PTP2 is found, the root, level 1 and level 2 look-ups can be bypassed, and a PTE can be immediately read from memory.

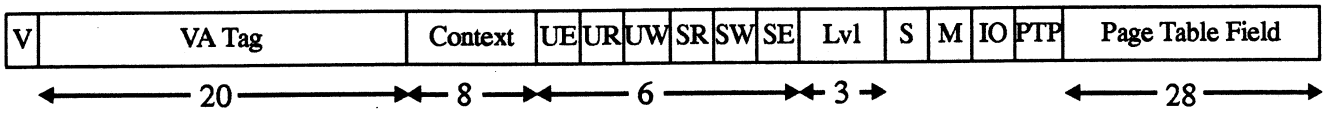
Figure 5.1 - Possible TLB Replacement



5.2.2 TLB Entry

An entry in the TLB has the following fields: a virtual address tag, a context tag, a PTE level field, and a page table field.

Figure 5.2 - TLB Entry



Field Definitions:

Valid (V) - This bit is used to indicate that the entry holds valid information.

Virtual Address Tag - The 20 bit virtual address tag represents the most significant 20 bits (VA[31:12]) of the virtual address being used when referencing PTE and IOPTE. VA[11:00] is the byte offset within a page. The address in this field is physical when

referencing PTPs with the least significant bits containing PA[26:08].

Context Tag - The 8 bit context tag comes from the value in the context register as written by memory management software when referencing PTEs. Both it and the virtual address tag must match the CXR and VA[31:12] in order to have a TLB hit. This field contains a physical address (PA[07:02]) when referencing PTPs. Note that for PTPs only 6 bits are used to hold PA[07:02], the two lower bits are always set to "0". This field is not used when referencing IOPTEs.

Prot - The 6 protection bits in each TLB entry represent the decoded ACC bits from the matching PTE, namely user Rd, Wr, Ex, and supervisor Rd, Wr, Ex. These bits are used to check for protection violations on entries that meet the TLB hit criteria.

Level - The 3 bit level field is used to enable the proper virtual tag match of region, and segment PTE's. IOPTE's and PTP's will have this field set to use Index 1, 2 and 3 (b'000').

Table 8 - Virtual Tag Match Criteria

Level	Match Criteria
111	None
011	Index1 (VA[31:24]
001	Index 1,2 (VA[31:18])
000	Index 1, 2, 3 (VA[31:12])

Supervisor (S) - This bit is used to disable the matching of the context field indicating that a page is a supervisor level (ACC=6 or 7).

Modified (M) - This bit is set to a one when the page is written.

IO Page Table Entry (IO) - This bit indicates that an IOPTE resides in this entry of the TLB.

Page Table Pointer (PTP) - This bit indicates that a PTP resides in this entry of the TLB. Note that all SRMMU flush types (except page) will flush all PTPs from the TLB.

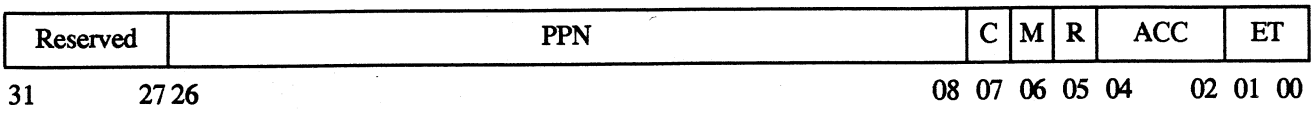
Page Table Field - The page table field can either be a Page Table Entry (PTE), a Page Table Pointer (PTP), or an IO Page Table

Entry (IOPTE). This field can be read and written using ASI 0x06.

5.2.3 Page Table Entry

A Page Table Entry (PTE) defines both the physical address of a page and its access permissions. A PTE is defined for SPARC reference MMUs as follows.

Figure 5.3 - Page Table Entry in Page Table



Field definitions:

Reserved (Rsvd) - Bits [31:27] should be written as zero, and will be read as zero.

Physical Page Number (PPN) - This field is the high order 19 bits ([30:12]) of the 31 bit physical address of the page. The PPN appears on PA[30:12] when a translation completes.

Cacheable (C) - When this bit is set to a one the page is cacheable by an instruction and/or data cache.

Modified (M) - This bit is set to a one when the page is written to.

Referenced (R) - This bit is set to a one when the page is accessed. All PTEs in the TLB have this bit set when the entry is loaded.

Access Permissions (ACC) - These bits indicate whether access to this page is allowed for the transaction being attempted. The Address Space Identifier (ASI) determines whether a given access is a data access or an instruction access, and whether the

access is being done by the user or supervisor. The field is defined as follows.

Table 9 - Page Table Access Permission

ACC	Access Mode	
	User	Supervisor
0	Read Only	Read Only
1	Read/Write	Read/Write
2	Read/Execute	Read/Execute
3	Rd/Wr/Execute	Rd/Wr/Execute
4	Execute Only	Execute Only
5	Read Only	Read/Write
6	No Access	Read/Execute
7	No Access	Rd/Wr/Execute

Entry Type (ET) - This field differentiates the entry types in the TLB. Note that the entry type is not kept in the TLB RAM. On a probe operation the ET field is derived from a combination of other bits. The bit definitions of the ET field follows:

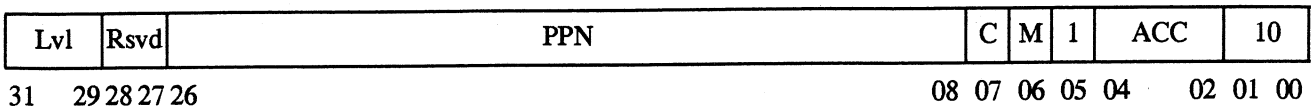
Table 10 - Page Table Entry Types

ET	Entry Type
0	Invalid
1	Page Table Pointer
2	Page Table Entry
3	Reserved in Page Tables.

"Invalid" means that the corresponding range of virtual addresses is not currently mapped to a physical address.

In the TLB RAM the PTE has the following format:

Figure 5.4 - Page Table Entry in TLB



Bits [28:27] are not implemented, should be written as zero, and will be read as zero.

Bit [05] is set to one by hardware indicating that every PTE in the TLB has been referenced.

Bits [01:00] are set to 2'b10 by hardware indicating the entry type (ET) of a PTE. These bits are not actually stored in the TLB rather are derived as a function of the PTP bit of the tag.

Bits[31:29] are set to indicate the page table level where the entry is to be found. The following table describes the possible encodings:

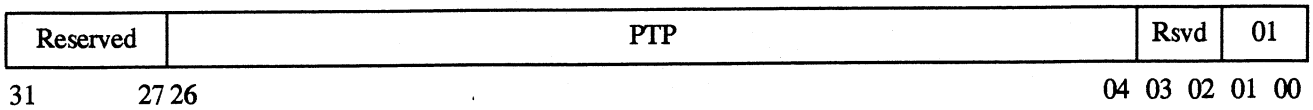
Table 11 - Page Table Entry Level in TLB

Lvl	Page Table Level
000	Level 0 (Root)
100	Level 1
110	Level 2
111	Level 3 or IOPTE

5.2.4 Page Table Pointer

A Page Table Pointer (PTP) contains the physical address of a page table and may be found in the Context Table, in a Level 1 Page Table, or in a Level 2 Page Table. Page Table Pointers are put into the TLB during tablewalks and removed from the TLB either by natural replacement (also during tablewalks) or by flushing the entire TLB. Note that the Level field in a PTP tag is always set to 0x7. A PTP is defined as follows:

Figure 5.5 - Page Table Pointer in Page Table



Field definitions:

Reserved (Rsvd) - Bits[31:30,03:02] should be written as zero, and will be read as zero.

Page Table Pointer (PTP) - The physical address of the base of a next level page table. The PTP appears on PA[30:08] during miss processing. The page table pointed to by a PTP must be aligned on a boundary equal to the size of the page table. Note that this is true of the context table at the root level also. The sizes of the tables are summarized as follows.

Table 12 - Size of Page Tables

Level	Size (Bytes)
Root	1024
1	1024
2	256
3	256

Entry Type (ET) - This field differentiates the entry types in the TLB. Note that the entry type is not kept in the TLB RAM. On a

probe operation the ET field is derived from a combination of other bits. The bit definitions of the ET field follows:

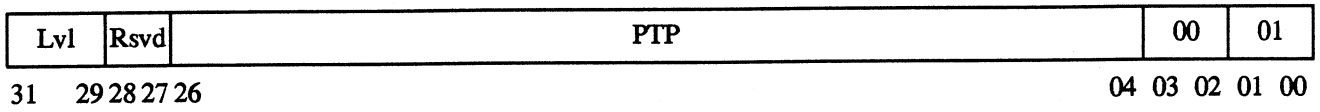
Table 13 - Page Table Entry Types

ET	Entry Type
0	Invalid
1	Page Table Pointer
2	Page Table Entry
3	Reserved

"Invalid" means that the corresponding range of virtual addresses is not currently mapped to a physical address.

In the TLB, a PTP has the following format:

Figure 5.6 - Page Table Pointer in TLB



Bits [28:27] are not implemented, should be written as zero, and will be read as zero.

Level (Lvl) - The level bits for a PTP indicate the level at which the PTP is found. The following table shows the possible level encodings.

Table 14 - Page Table Entry Types

Lvl[2:0]	PTP level
000	Level-0 (root)
100	Level-1
110	Level-2

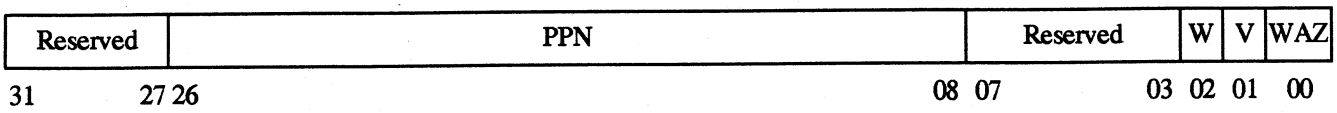
Bits [03:02] are set to zero by hardware and are unused.

Bits [01:00] are set to 2'b01 by hardware indicating the entry type (ET) of a PTP. These bits are not actually stored in the TLB rather are derived as a function of the PTP bit of the tag.

5.2.5 IO MMU Page Table Entry

An IO Page Table Entry (IOPTE) defines both the physical address of a page and its access permissions. Note that the Level field in a IOPTE tag is always set to 0x7 and the Supervisor bit is set to 0x0. An IOPTE is defined as follows.

Figure 5.7 - IO Page Table Entry in Page Table



Field definitions:

Reserved (Rsvd) - Bits [31:27] are not implemented, should be written as zero, and will be read as zero. Bits [07:03] should also be written as zero, and will be read as zero.

Physical Page Number (PPN) - This field is the high order 19 bits of the 31 bit physical address of the page. The PPN appears on PA[30:12] when a translation completes. This address is concatenated with VA[11:00] to provide the entire translated address.

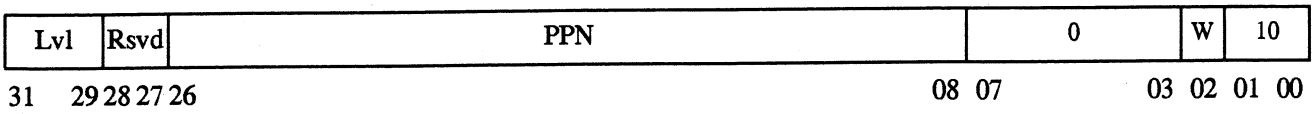
Writable (W) - When this bit is set to a one both reads and writes to the page are allowed. When this bit is zero only reads are allowed.

Valid (V) - This bit is set to a one when the IOPTE is valid.

Write As Zero (WAZ) - This bit is to be written as zero in the memory io pagetable by software.

In the TLB an IOPTE has the following format:

Figure 5.8 - IO Page Table Entry in TLB



Bits [28:27] are not implemented, should be written as zero, and will be read as zero.

Bits [07:03] are set to zero by hardware. Bit[05] is used to distinguish between PTEs (set to 1) and IOPTEs (set to 0). Bits[07:06,04:03] are unused.

Bits [01:00] are set to 2'b10 by hardware indicating a valid IOPTE. These bits are not actually stored in the TLB.

5.3 Address Space decodes

The physical address space for microSPARC-II is decoded into eight address spaces, based on the upper three bits of the physical address(pa[30:28]). The following table defines the address spaces and their decodes:

Table 15 - Virtual Tag Match Criteria

PA[30:28]	Address Space
000	Main Memory Space
001	Control Space (Sun-4M system registers)
010	Local Graphics frame buffer space
011	I/O Space (SBus slave select 0)
100	I/O Space (SBus slave select 1)
101	I/O Space (SBus slave select 2)
110	I/O Space (SBus slave select 3)
111	I/O Space (SBus slave select 4)

5.4 CPU TLB Lookup

A virtual address to be translated by the MMU is compared to each entry in the TLB. During the TLB lookup the value of the Level field specifies which index fields are required to match the TLB virtual tag as follows:

Table 16 - Virtual Tag Match Criteria

Level	Match Criteria
111	None
011	Index1 (VA[31:24]
001	Index 1,2 (VA[31:18])
000	Index 1, 2, 3 (VA[31:12])

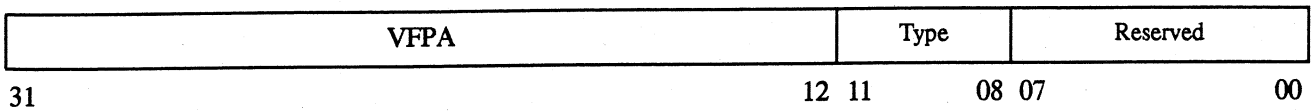
In addition to the virtual tag match, context matching of a PTE is required for all user page references (ACC is 0 to 5) when made by either user or supervisor (ASI = 0x8-0xB). Context matching is not required for a supervisor page reference (ACC is 6 or 7) when made by a supervisor (ASI = 0x9 or 0xB). This case takes advantage of the Supervisor bit in the TLB tag. Note that user references (ASI = 0x8 or 0xA) to supervisor pages (ACC is 6 or 7) result in address exceptions.

Note that the TLB ignores access level checking during probe operations. The most significant Level field bit is used as a Valid bit for the TLB. This means that root level PTEs are not supported.

5.5 CPU TLB Flush and Probe Operations

The flush operation allows software invalidation of TLB entries. TLB entries are flushed by using a store alternate instruction. The probe operation allows testing the TLB and page tables for a PTE corresponding to a virtual address. TLB entries are probed by using a load alternate instruction. The ASI value 0x3 is used to invalidate or probe entries in the TLB. In an alternate address space used for probing and flushing the address is composed as follows:

Figure 5.9 - CPU TLB Flush or Probe Address Format



Field Definitions:

Virtual Flush or Probe Address (VFPA) - This field is the address that is used as the match criterion for the flush or probe operations into TLB. Depending on the type of flush or probe not all 20 bits are significant. Note that context flush uses the current context id as defined in the context register.

Type - This field specifies the extent of the flush or the level of the entry probed.

Reserved - These bits are ignored. They should be set to zero.

5.5.1 CPU TLB Flush

The flush operation must remove the PTEs and PTPs from the TLB that match the type criteria as follows:

Table 17 - TLB Entry Flushing

Type	Flush	PTE Match Criteria
0	Page	((ACC ≥ 6) OR CID match) AND VA[31:12] match
1	Segment	((ACC ≥ 6) OR CID match) AND VA[31:18] match
2	Region	((ACC ≥ 6) OR CID match) AND VA[31:24] match
3	Context	(ACC ≥ 6) OR CID match
4	Entire	None (Entire TLB Flush)
5 to F	Reserved	-

Page flush only removes matching PTEs from the TLB. All other flushes will remove matching PTEs and all PTPs from the TLB. CPU flush context operations will flush PTEs that match the current context, and all PTEs that have the S(Supervisor) bit set in their tags, If the CPU

is running with virtual PTPs enabled, all virtually tagged PTPs are flushed for any occurence of flush context, region, or segment. Flush operations to types 5-F are reserved and will not affect the TLB

5.5.2 CPU TLB Probe

The probe operation returns either a PTE from a page table in main memory or the TLB or it returns a zero if there is an invalid address or translation error while searching for the entry implied by the probe. If there is an error, a zero is returned for data. The reserved probe types (0x5-0xF) return an undefined value. A type 4 probe (entire) brings the accessed PTE and any PTPs that were needed into the TLB. If the PTE was not already there the referenced bit is updated. Probe types 0-3 affect one entry of the TLB which is invalidated at the end of the probe operation.

The value returned by a probe operation is specified in the following table. For a given probe type, the table is read left-to-right. "0" indicates that a zero is returned. "X" indicates that the page table entry itself is returned, and "->" indicates that the next-level page table entry is examined.

Table 18 -Return Value for MMU Probes

Type	Level - 0 Entry Type				Level - 1 Entry Type				Level - 2 Entry Type				Level - 3 Entry Type				Mem Err
	pte	res	inv	ptp	pte	res	inv	ptp	pte	res	inv	ptp	pte	res	inv	ptp	
0(page)	0	0	0	→	0	0	0	→	0	0	0	→	X	0	X	0	0
1(seg)	0	0	0	→	0	0	0	→	X	0	0	X	--	--	--	--	0
2(reg)	0	0	0	→	X	0	X	X	--	--	--	--	--	--	--	--	0
3(ctx)	X	0	X	X	--	--	--	--	--	--	--	--	--	--	--	--	0
4(entire)	X	0	0	→	X	0	0	→	X	0	0	→	X	0	0	0	0
5-0xF	(undefined)																

5.6 Processor MMU Registers

The Processor Control Register (CR) contains general CPU control and status flags. The current context identifier is stored in the Context Register (CXR), and a pointer to the base of the context table in memory

is stored in the Context Table Pointer Register (CTPR). If an MMU fault occurs on a CPU initiated transaction the address causing the fault is placed in the Synchronous Fault Address Register (SFAR) and the cause of the fault can be determined from the contents of the Synchronous Fault Status Register (SFSR). The TLB Replacement Control Register is used to control which TLB entries are to be replaced next. All of these internal MMU registers can be accessed directly by the processor through alternate address space word accesses with an ASI value 0x4. The address map for these registers follows.

Table 19 - Address Map for MMU Registers

VA[12:08]	Register
00	Processor Control Register
01	Context Table Pointer Register
02	Context Register
03	Synchronous Fault Status Register
04	Synchronous Fault Address Register
05-0F	Reserved
10	TLB Replacement Control Register
11-12	Reserved
13	Synchronous Fault Status Register**
14	Synchronous Fault Address Register**
15-1F	Reserved

VA bits [31:13] are zero. VA bits [07:00] are ignored and should be set to zero by software. The use of a second access mode for the Synchronous Fault registers is provided as a diagnostic function (VA[12:08] = 0x13, 0x14). See register description for details.

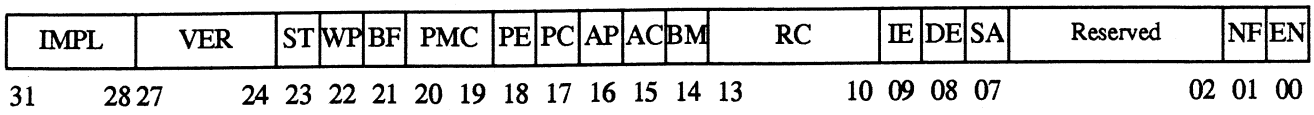
** Registers are cleared on read when this address is used.

5.6.1 Processor Control Register

The Processor Control Register contains control and status bits for the microSPARC-II processor. The BM, IE, DE, and EN bits receive both the SBus reset (normal reset) and watchdog resets (BM is set, IE, DE, and EN are reset). It is highly recommended that sta's to the PCR are

immediately followed by 10 NOP instructions to keep the machine in a very consistent state. The PCR is defined as follows:

Figure 5.10 - Processor Control Register



Field Definitions:

Reserved (Rsvd/Rsv) - Bits [22:21,16,07:02] are unimplemented, should be written as zero and will be read as zero.

Implementation (IMPL) - The implementation number of this SPARC Reference MMU. This field is hardwired to 0x0 and read only.

Version (VER) - The version number of this SPARC Reference MMU. This field is hardwired to 0x4 read only.

Software Tablewalk enable (ST) - This bit enables the instruction_access_MMU_miss and data_access_MMU_miss traps for instruction and data tablewalking respectively for tablewalks to be done by software.

Watch point enable - This bit enables the watch point trap. When set, this bit will enable the Watch Point Trap logic within the MMU logic.

Branch folding - This bit enables IU branch folding operation. When set this bit will enable the branch folding feature in the IU logic.

Page Mode Control(PMC) - This bit enables the Page mode operation of the MMU/MEMIF interface. When this bit is set The MMU's page mode registers will track the usage of pages in memory to take advantage of page mode access to the DRAM when possible. Bit[19] controls page hit register 0, and Bit[20] controls page hit register 1. These bits are cleared on reset.

Local Graphics Page Mode Control(AP) - This bit enables the Page mode operation of the Local Graphics interface. When this bit is set, the MMU's page mode registers will track the usage of pages in Local Graphics space to take advantage of page mode access to Local Graphics when possible. This bit is cleared on reset.

Refresh Control (RC) - These 4 bits control the DRAM refresh rate of the system. Normal 70MHz operation would require a 0x3 value. The RC field is defined as follows:

Table 20 - Memory Refresher Control Definition

RFR_CNTL	Refresh Interval
0000	Every 128 MCLKs (down to 8.6Mhz)
0001	No Refresh
0010	Every 704 MCLKs (down to 48Mhz)
0011	Every 896 MCLKs (down to 60Mhz)
0100	Every 1216 MCLKs (to 83 Mhz)
0101	Every 5120 MCLKs (low refresh)
0110	Every 1408 MClks (down to 100 Mhz)
0111	Every 1792 MClks (down to 125 Mhz)
1xxx	Self Refresh DRAMs

Parity Control (PC) - This bit controls the generation of parity (and checking on memory reads) in the memory interface as follows:

Table 21 - Parity Control Definition

PC	Meaning
0	Even Parity
1	Odd Parity

Boot Mode (BM) - This bit is set by both SBus reset and watchdog reset and must be cleared for normal operation.

Parity Enable (PE) - When set to one this bit enables word parity checking for all data entering the processor over the memory bus.

Instruction Cache Enable (IE) - The instruction cache is enabled when this bit is set to a one. When zero, all references miss the cache. This bit is reset by both SBus reset and watchdog reset.

Data Cache Enable (DE) - The data cache is enabled when this bit is set to a one. When zero, all references miss the cache. This bit is reset by both SBus reset and watchdog reset.

Store Allocate (SA) - When set, this bit enables user store misses to be run in allocate mode. This means that if the page has been mapped as cachable, the MMU will signal the D-cache that a line fill must be done to satisfy the store miss. If the bit is cleared the MMU will disregard the page mapping information, and signal the D-cache that no line fill is required. The effect is that subsequent accesses to this data will also "miss" in the D-cache if no allocate was done during the store miss. However, the amount of time the CPU is stalled will be reduced when the allocate is not done. In either case the store data is placed in the store buffer, and subsequently to memory. No cache fill is done regardless of the lines cacheability. The bit has no affect on Supervisor store misses. All Supervisor store misses are done in no-allocate mode. The following table shows the possible settings:

Table 22 - Store Allocate Setting

SA	User ST miss	Sys ST miss
0	NO Allocate	NO Allocate
1	Allocate	NO allocate

No Fault bit (NF) - When the NF bit is set, any access to an ASI other than 8 or 9, that causes a fault will be captured in the FSR and FAR, but no fault is generated to the processor. Faults resulting from access to ASI 8 or 9 will be handled as normal regardless of the setting of this bit. Normal operation occurs while this bit is cleared.

MMU Enable (EN) - When this bit is set to a one the MMU is enabled and translation occurs normally. When this bit is not set the physical address is forced to the 31 least significant bits of the virtual address. This bit is reset by both SBus reset and watchdog reset.

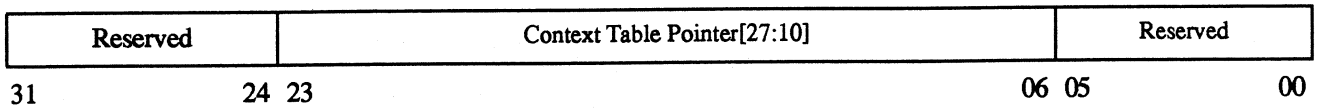
Alternate Cacheability (AC) - When set, this bit specifies that the caches are enabled by the IE and DE bits even with the MMU disabled. When not set, the caches are disabled when the MMU is disabled. This should not be used during boot mode accesses (or other instruction access to an SBus device.). The access

privilege associated with memory operations done under alternate cacheability mode is hardwired to acc = 0x011 (User R/W/E and Sys R/W?E). Alternate Cacheability is a diagnostic feature that allows the caches to be enabled by the IE and DE bits even with the MMU disabled. When not set, the caches are disabled when the MMU is disabled. This should not be used during boot mode accesses (or other instruction accesses to an SBus device). Specifically, having the MMU off, instruction cache on, alternate cacheability on and an SBus instruction access can cause indeterminate data to be put into the instruction cache. Instruction accesses work fine with alternate cacheability when the accesses are to main memory space.

5.6.2 Context Table Pointer Register

The Context Table Pointer Register (CTPR) contains the base of the Context table. It is defined as follows.

Figure 5.11 - Context Table Pointer Register

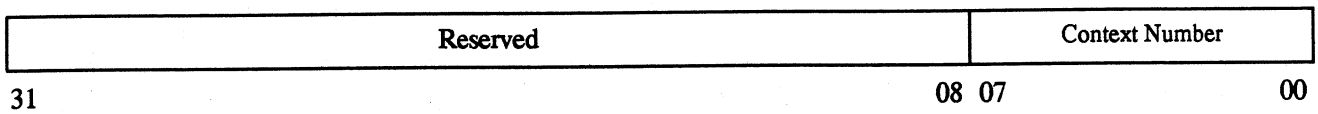


The Context Table Pointer is 18 bits wide. The reserved fields are unimplemented, should be written as zero, and read as a zero.

5.6.3 Context Register

The Context Register (CXR) is used as an index into the Context table. It is defined as follows.

Figure 5.12 - Context Register



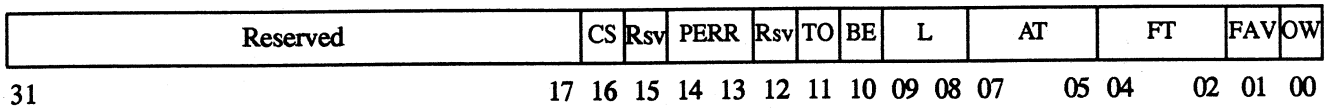
The Context Register defines which virtual address space is considered the "current" address space. Subsequent accesses to memory through the MMU are translated for the current address space. This continues

until the CXR is changed. The physical address of the root pointer is obtained by taking bits [23:06] from the CTPR to form mm_pa[27:10] and bits [07:00] from the CXR to form mm_pa[09:02]. mm_pa[30:28,01:00] are zero. Bits [31:08] of the CXR are unimplemented, should be written as zero, and read as a zero.

5.6.4 Synchronous Fault Status Register

The Synchronous Fault Status Register (SFSR) provides information on exceptions (faults) issued by the MMU during CPU type transactions. There are three types of faults: instruction access faults, data access faults, and translation table access faults. If another instruction access fault occurs before the fault status of a previous instruction access fault has been read by the IU, the latest fault status is written into the SFSR and the OW bit is set. If multiple data access faults occur only the status of the one taken by the IU is latched into the SFSR (and address in the SFAR). If data fault status overwrites previous instruction fault status the OW bit is cleared since the fault status is represented correctly. An instruction access fault does not overwrite a data access fault. If a translation table access fault overwrites a previous instruction or data access fault the OW bit is cleared. An instruction access or data fault does not overwrite a translation table access fault. Reading the SFSR using ASI 0x4 and VA[12:08]= 0x03 clears it.. However, reading the SFSR with VA[12:08]=0x13 does not clear it. Writes to the SFSR using ASI 0x4 and VA[12:08]=0x03 have no effect while writes using VA[12:08]=0x13 update the register. The SFSR is only guaranteed to be valid after an exception is actually signalled. In other words, it may not be valid if there is no exception.

Figure 5.13 - Synchronous Fault Status Register



Field Definitions:

Reserved (Rsvd) - Bits [31:17,15,12] are not implemented, should be written as zero, and read as zero.

Control Space Error (CS) - This bit is asserted on the following conditions: [1] invalid ASI space, [2] invalid ASI size, [3] invalid VA field in valid ASI space and [4] invalid ASI operation (for example a swap instruction to an ASI other than

0x8-0xB,0x20). Note that the AT field is not valid on Control Space Errors.

Parity Error (PERR) - The Parity Error[1:0] bits are set for external memory bus parity errors on the even and odd words respectively from memory.

SBus Time Out (TO) - An SBus Time Out resulted from a CPU initiated read transaction. No SBus slave responded with an acknowledge within 256 SBus cycles.

SBus Bus Error (BE) - An error indication was returned from an SBus slave on a CPU initiated read transaction. This may have been either an error acknowledgment or a late error.

Level (L) - The Level field is set to the page table level of the entry which caused the fault. If an error occurs while fetching a page table (either a PTP or PTE) this field records the page table level for the entry. The level field is defined as follows.

Table 23 - SFSR Level Field

L	Level
0	Entry in Context Table
1	Entry in Level 1 Page Table
2	Entry in Level 2 Page Table
3	Entry in Level 3 Page Table

Access Type (AT) - The Access Type field defines the type of access which caused the fault. Loads and Stores to user/supervisor instruction space can be caused by load/store alternate instructions with ASI = 0x8-0xB. The AT field is defined as follows. Note that this field is not valid on Control Space Errors.

Table 24 - SFSR Access Type Field

AT	Access Type
0	Load from User Data Space
1	Load from Supervisor Data Space
2	Load/Execute from User Instruction Space
3	Load/Execute from Supervisor Instruction Space

Table 24 - SFSR Access Type Field

AT	Access Type
4	Store to User Data Space
5	Store to Supervisor Data Space
6	Store to User Instruction Space
7	Store to Supervisor Instruction Space

Fault Type (FT) - The Fault Type field defines the type of the current fault. The FT field is defined as follows.

Table 25 - SFSR Fault Type Field

FT	Fault Type
0	None
1	Invalid Address Error
2	Protection Error
3	Privilege Violation Error
4	Translation Error
5	Access Bus Error
6	Internal Error
7	Reserved

Invalid address errors, protection errors, and privilege violation errors depend on the AT field of the SFSR and the ACC field of the corresponding PTE. The errors are set as follows

Table 26 - Setting of SFSR Fault Type Code

AT	FT Code								
	PTE[V]=0	PTE[V]=1(ACC)							
		0	1	2	3	4	5	6	7
0	1	-	-	-	-	2	-	3	3
1	1	-	-	-	-	2	-	-	-

Table 26 - Setting of SFSR Fault Type Code

AT	FT Code								
	PTE[V]=0	PTE[V]=1(ACC)							
		0	1	2	3	4	5	6	7
2	1	2	2	-	-	-	2	3	3
3	1	2	2	-	-	-	2	-	-
4	1	2	-	2	-	2	2	3	3
5	1	2	-	2	-	2	-	2	-
6	1	2	2	2	-	2	2	3	3
7	1	2	2	2	-	2	2	2	-

A translation error code (FT=4) is set when a SFSR PE type error occurs while the MMU is fetching an entry from a page table, a PTP is found in a level 3 page table, or a PTE has ET=3. The L field records the page table level at which the error occurred. The PE field records the word(s) having a parity error, if any. The protection error code (FT=2) is set if an access is attempted that is inconsistent with the protection attributes of the corresponding PTE. The privilege error code (FT=3) is set when a user program attempts to access a supervisor only page. An access bus error code (FT=5) is set when the SFSR PE field gets set on a memory operation that was not a table walk, or on a synchronously generated SBus error acknowledge or time out. Additionally, this error code is also set on an alternate space access to an unimplemented or reserved ASI or the memory access is using a size prohibited by the particular type of ASI. If multiple errors occur on a single access the highest priority fault is recorded in the FT field (see below).If a single access causes multiple errors, the fault type is recognized in the following priority.

Table 27 - Priority of Fault Types on Single Access

Priority	Fault Type
1	Internal Error
2	Translation Error
3	Invalid Address Error

Table 27 - Priority of Fault Types on Single Access

Priority	Fault Type
4	Privilege Violation Error
5	Protection Error

Fault Address Valid (FAV) - The Fault Address Valid bit is set if the contents of the Synchronous Fault Address Register (SFAR) are valid. The SFAR is valid for data exceptions and data errors.

Overwrite (OW) - The Overwrite bit is set if the SFSR has been written more than once to indicate that previous status has been lost since the last time it was read.

Table 28 - Overwrite Operations

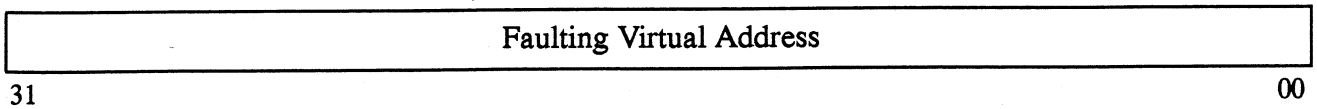
Pending Error	New error	OW Status	Action Signalled
Translation Error	Translation Error	Set	Translation Error
Translation Error	Data Access Exception	Unchanged	Data Access Exception
Translation Error	Instruction Access Exception	Unchanged	Instruction Access Exception
Data Access Exception	Translation Error	Clear	Translation Error
Data Access Exception	Data Access Exception	Set	Data Access Exception
Data Access Exception	Instruction Access Exception	Unchanged	Instruction Access Exception
Instruction Access Exception	Translation Error	Clear	Translation Error
Instruction Access Exception	Data Access Exception	Clear	Data Access Exception
Instruction Access Exception	Instruction Access Exception	Set	Instruction Access Exception

5.6.5 Synchronous Fault Address Register

The Synchronous Fault Address Register (SFAR) records the 32 bit virtual address of any data fault or translation reported in the SFSR. The SFAR is overwritten according to the same policy as the SFSR on data faults. Reading the SFAR using ASI 0x4 and VA[12:08] 0x04 clears it. Using VA[12:08] 0x14 to read the SFSR does not clear it. Writes to the SFAR using ASI 0x4 and VA[12:08] 0x04 have no effect while writes using VA[12:08] 0x14 update the register. Note that the SFAR should

always be read before the SFSR to insure that a valid address is returned. The structure of this register is as follows.

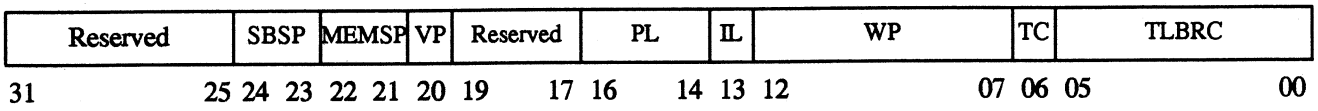
Figure 5.14 - Synchronous Fault Address Register



5.6.6 TLB Replacement Control Register

The TLB Replacement Control Register (TRCR) contains the TLB Replacement Counter and counter disable bit. The TRCR can be read and written using alternate load/store (LDA and STA) at ASI 0x4 with VA[12:08]=0x10. It is defined as follows.

Figure 5.15 - TLB Replacement Control Register



Field Definitions:

Reserved - Bits [31:21,19:17] are unimplemented, should be written as zero and will be read as zero.

SBus Speed (SBSP) - These bits are used to indicate the divide by speed select used to generate the SBus clock from the internal CPU clock. The value is encoded as shown in the following table:.

Table 29 - SBus Speed Select

SBSP	SBus clock divide by
0	2
1	3
2	4
3	5

Memory Speed (MEMSP) - These bits are used to indicate the speed select being used for the DRAM memory interface. The values are encoded as shown in the following table:.

Table 30 - Memory Speed Select

MEMSP	Fault Type
0	0 - 70 Mhz
1	1 - 85 Mhz
2	10 - 100 Mhz
3	11 - 125 Mhz

Virtually tagged Ptps (VP): This bit is used to enable the tagging of level2 PTPs, in the TLB, with virtual tags instead of physical tags. When a table walk is started we will check for a virtually tagged level2 PTP, before checking for a root level PTP. This check will not make the tablewalk any longer than usual. If a VPTP2 is found, the tablewalk goes directly to the level3 pte lookup. The TLB should be flushed after setting, or resetting this bit to avoid mixing physically tagged level2 PTPs and virtually tagged level2 PTPs.

TLB Replacement Counter Disable (TC) - The TLBRC will not increment when this bit is set.

TLB Replacement Counter (TRC) - This is a 6 bit modulo 64 counter which is incremented by one during each CPU clock cycle to point to one of the TLB entries unless the TC bit is set. When a TLB miss occurs, the counter value is used to address the entry to be replaced.

Wrap around Point (WP) - This 6 bit is to set a wrap around point for TLB replacement.

I/O PTE Lock (IL) - This bit is used to enable I/O PTE location limits. When this bit is set I/O PTEs are only placed into TLB locations 48-63. The replacement within these bounds is still pseudo-random. If entries have been locked down via a WP setting, these entries should be manually placed above entry 48, and the WP set to 47-(63-old WP setting).

PTP Lock (PL) - This bit is used to enable PTP location limits. When this bit is set, PTP placement in the TLB will be limited to entries

0-2. Bit[16] locks PTPs for level 2, Bit[15] locks PTPs for level 1, and Bit[14] locks PTPs for level 0. When PTP lock is used the wrap point should not be set to a value less than 0x5. The MMU will try to use locations 3, 4, and 5 as alternate PTE stores when 0, 1, and 2 are reserved for PTPs. This use of locations 3, 4, and 5 is done without regard for the current setting of the WP.

5.7 IO MMU Registers

The IO MMU Control Register (IOCR) contains IO MMU control and status flags. The IO MMU Base Address Register (IOBAR) defines the base address of the IO PTE Table in memory. The SBus Slot Configuration Registers (SSCR[0:4]) provides information about the slave device in the SBus slots. If a parity error occurs on an IO initiated transaction the physical address causing the fault is placed in the Asynchronous Fault Address Register (AFAR) and the cause of the fault can be determined from the contents of the Asynchronous Fault Status Register (AFSR). A DMA parity error will result in asserting the level 15 interrupt output (to be fed back to the IU externally as an interrupt) and the assertion of an error acknowledge to the SBC so it can return an SBus error acknowledge to the device that initiated the transaction. The Sun4m architecture does not require error checking, of operation size, for DVMA accesses to Control Space. microSPARC-II does not thoroughly check for size errors on DVMA accesses to control space. IOPTE entries may be flushed from the TLB by doing writes to the Address Flush Register (AFR). This register is write only. All of these internal MMU registers can be accessed directly by software using SBus and IO MMU Control Space accesses with PA[30:24]=0x10. Also, the entire TLB can be flushed using a control space access. The SBus and IOMMU Control Space address map follows.

Table 30 - SBus, IO MMU, and Perf Counter Control Space

PA[30:00]	Device	R/W
1000 0000	IO MMU Control Register	R/W
1000 0004	IO MMU Base Address Register	R/W
1000 0014	Flush All TLB Entries	W
1000 0018	Address Flush Register	W
1000 1000	Asynchronous Fault Status Reg	R/W
1000 1004	Asynchronous Fault Address Reg	R/W

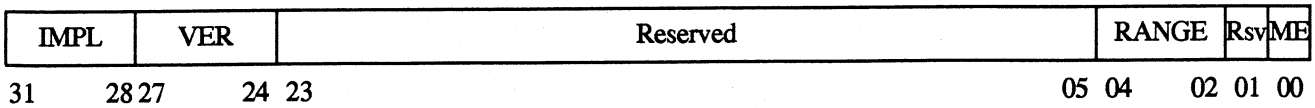
Table 30 - SBus, IO MMU, and Perf Counter Control Space

PA[30:00]	Device	R/W
1000 1010	SBUS Slot Configuration Register0	R/W
1000 1014	SBUS Slot Configuration Register1	R/W
1000 1018	SBUS Slot Configuration Register2	R/W
1000 101C	SBUS Slot Configuration Register3	R/W
1000 1020	SBUS Slot Configuration Register4	R/W
1000 1050	Memory Fault Status Register	R/W
1000 1054	Memory Fault Address Register	R/W
1000 2000	MID Register	R/W
1000 3000	Trigger A Enables Register	R/W
1000 3004	Trigger B Enables Register	R/W
1000 3008	Assertion Control Register	R/W
1000 300C	MMU Breakpoint Control Register	R/W
1000 3010	Performance Counter A	R/W
1000 3014	Performance Counter B	R/W
1000 3018	VA Mask Register	R/W
1000 301C	VA Compare Register	R/W
1000 4000	Local Graphics Queue Level	W
1000 6000	Local Graphics Queue Level	R/O
1000 7000	Local Graphics Queue Status	R/O

5.7.1 IO MMU Control Register

The IO MMU Control Register (IOCR) contains control and status bits for the IO MMU. This register can be accessed using SBus and IO MMU Control Space (0x10000000). The IOCR is defined as follows:

Figure 5.16 - IO Control Register



Field definitions:

Implementation (IMPL) - The implementation number of this IO MMU. This field is hardwired to 0x0 and read only.

Version (VER) - The version number of this IO MMU. This field is hardwired to 0x4 and read only.

Reserved (Rsv) - Bits [23:05,01] are not implemented, should be written as zero, and will be read as zero.

RANGE - This field defines the virtual address range for DVMA. Specifically, the translatable limit is defined to be 16MB*2**<RANGE>. All VA bits above this limit must be set to one for an address to be valid. For example, if RANGE=2 then 64MB of virtual address are supported, and valid DVMA virtual addresses range from 0xFC000000 to 0xFFFFFFFF. Any access using a DVMA virtual address that is out of that range will receive an SBus error acknowledge. The only exception involves slots that have Bypass Enabled. The following table shows how the physical address of an IO MMU page table entry

is generated:

Table 31 - IO MMU Page Table Address Generation

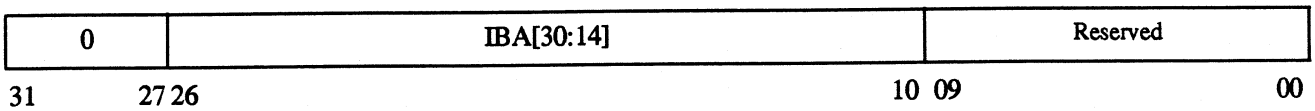
Range	Limit	Physical Address[30:00]
0	16MB	IBAR[26:10], IOVA[23:12],b'00'
1	32MB	IBAR[26:11], IOVA[24:12],b'00'
2	64MB	IBAR[26:12], IOVA[25:12],b'00'
3	128MB	IBAR[26:13], IOVA[26:12],b'00'
4	256MB	IBAR[26:14], IOVA[27:12],b'00'
5	512MB	IBAR[26:15], IOVA[28:12],b'00'
6	1GB	IBAR[26:16], IOVA[29:12],b'00'
7	2GB	IBAR[26:17], IOVA[30:12],b'00'

IO MMU Enable (ME) - IO MMU translation is enabled when this bit is set.

5.7.2 IO MMU Base Address Register

The IO MMU Base Address Register (IBAR) defines the base address of the IO Page Table. This register can be accessed using SBus and IO MMU Control Space (0x10000004). The IBAR is defined as follows.

Figure 5.17 - IO MMU Base Address Register



Field definitions:

Reserved (Rsvd) - Bits [31:27,09:00] are not implemented, should be written as zero, and will be read as zero.

IO MMU Base Address (IBA) - When the IO MMU is enabled and the access translation misses the TLB, IBA is used as the base address for the (<RANGE/1024>)byte-aligned IO MMU Page Table.

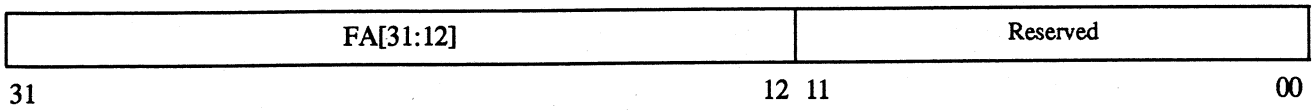
5.7.3 IOMMU Flush All TLB Entries

All I/O TLB entries are flushed by writing to control space address PA=0x10000014. This address should not be read since the output of the TLB is unknown during a flash clear operation.

5.7.4 IOMMU Address Flush Register

Individual IOPTE entries may be flushed from the TLB by doing writes to the Address Flush Register at PA=0x10000018 with the following format. The Address Flush Register is defined as follows.

Figure 5.18 - IOPTE Address Based Flush Format



Field definitions:

Reserved (Rsv) - Bits [11:00] are not implemented and should be written as zero.

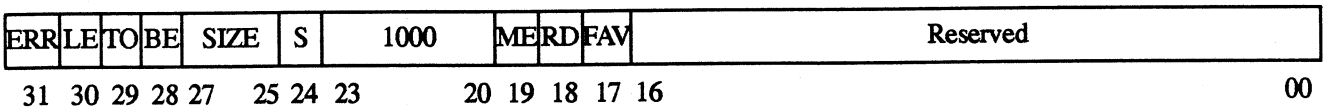
Flush Address (FA) - The virtual page address of the IOPTE entry to be flushed.

Note that a register is not actually implemented to perform this function.

5.7.5 Asynchronous Fault Status Register

The Asynchronous Fault Status Register (AFSR) provides information on asynchronous faults during IO initiated transactions and CPU write operations. This register is accessed using SBus and IO MMU Control Space (0x10001000). A hardware lock is used to ensure that this register does not change while being read. Reading this register unlocks it.

Figure 5.19 - Asynchronous Fault Status Register



Field Definitions:

Reserved (Rsvd) - Bits [23:20,16:00]. Bits [23:20] are forced to 0x1000. Bits [16:00] are not implemented, should be written as zero, and read as zero.

Summary Error Bit (ERR) - One or more of LE, TO, or BE is asserted.

Late Error (LE) - The SBus reported an error after the transaction was done.

Time Out (TO) - An SBus write access timed out. In a microSPARC-II based system this takes 256 SBus clks.

Bus Error (BE) - An SBus write access received an error acknowledge.

Size (SIZE) - SBus size of error transaction.

Supervisor (S) - CPU was in Supervisor mode when error occurred.

Multiple Error (ME) - At least one other error was detected after the one shown.

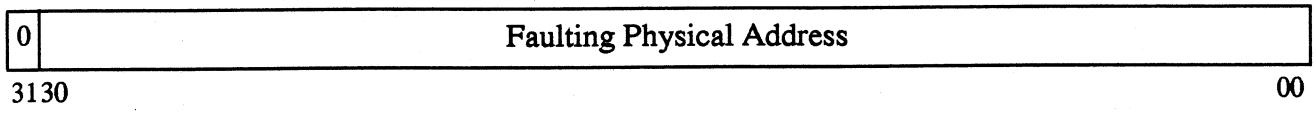
Read Operation (RD) - The error occurred during a read operation.

Fault Address Valid (FAV) - The address contained in the AFAR is accurate and can be used in conjunction with the status in AFSR. The only time the AFAR will be invalid is on an SBus late error in which the second processor IO operation has already been requested and is queued up in the SBC.

5.7.6 Asynchronous Fault Address Register

The Asynchronous Fault Address Register (AFAR) records the 31 bit physical address that caused the fault. This register is accessed using SBus and IO MMU Control Space (0x10001004). Bit [31] should be written as zero and will be read as zero. A hardware lock is used to insure that this register does not change while being read. Reading the AFSR unlocks the AFAR. The structure of this register is as follows.

Figure 5.20 - Asynchronous Fault Address Register

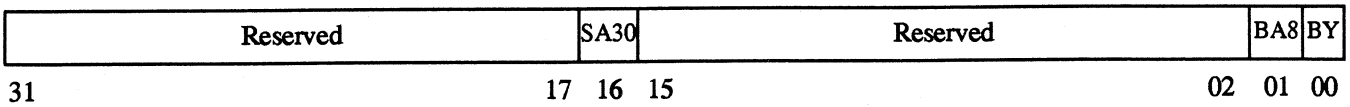


Note that bit 31 is unimplemented, should be written as zero, and will be read as zero. Also, this register is only held when an error is reflected in the AFSR.

5.7.7 SBus Slot Configuration Registers

The SBus Slot Configuration Registers (SSCR[0:4]) provide information about the slave device in SBus slots, and is also used for IO MMU bypass management for that slot. These registers can be accessed using SBus and IO MMU Control Space (0x10001010, 0x10001014, 0x10001018, 0x1000101C and 0x10001020 respectively). The BR/BG[5] and BR/BG[4] share SSCR4. They must be NON-BYPASS SBus device since the BY bits are hard-wired. (BY and SA30 bits are not used). Slaves sharing SSCR4 need external decoding of one more PA bit (e.g. PA[27], sbae[5:0] can also be used as a qualifier). If one slave does not support 8 byte burst the other must not support 8 byte burst(BA8 are used for both of slaves sharing SSCR4). The SSCR is defined as follows:

Figure 5.21 - SBUS Slot Configuration Register



Field definitions:

Reserved - Bits [31:17,15:03] are not implemented, should be written as zero, and will be read as zero.

Segment Address Bit 30 (SA30) - This bit provides PA[30] when IO MMU bypass is used.

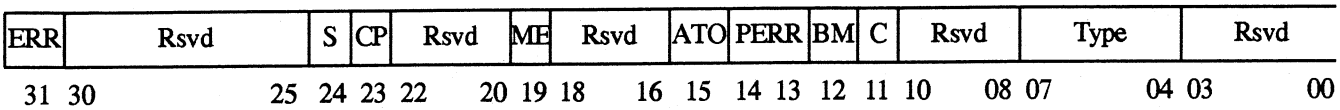
BA8 - Slave supports 8 byte bursts.

IO MMU Bypass (BY) - When this bit is set the MMU is bypassed and the virtual addresses from this slave are treated as physical when sb_ioa[31:30]=00. mm_pa[30] is given by the SA30 field and mm_pa[29:00] is defined as sb_ioa[29:00].

5.7.8 Memory Fault Status Register

The Memory Fault Status Register (MFSR) provides information on parity faults. This register is accessed using SBus and IO MMU Control Space (0x10001050). This register is loaded on every request to memory unless it is locked. A hardware lock is used to ensure that this register does not change while being read if there was an error condition. Reading this register allows it to begin loading once again.

Figure 5.22 - Memory Fault Status Register



Field Definitions:

Reserved (Rsvd) - Bits [30:25,22:20,18:15,10:08,03:00] are not implemented, should be written as zero, and read as zero.

Summary Error Bit (ERR) - One or more of PERR[1] or PERR[0] is asserted.

Supervisor (S) - CPU was in Supervisor mode when error occurred.

CPU Transaction(CP) - CPU initiate the transaction that resulted in the parity error.

Multiple Error (ME) - At least one other error was detected after the one shown.

Parity Error[1:0] (PERR) - These bits are set on external memory parity errors for the even and odd words (respectively) from memory. Parity errors can result from CPU or IO initiated memory reads and byte or halfword (8 or 16 bit) write operations (which result in read-modify-writes).

Local Graphics Timeout (ATO) - This bit is used to indicate that a time out has occurred for the current Local Graphics operation.

Boot Mode (BM) - This bit indicates that the error occurred while the PCR was indicating Boot Mode.

Cacheable (C) - Address of error was mapped cacheable. In CPU initiated transactions this bit was from the C bit of the PTE, otherwise it is set to zero.

Memory Request Type (Type[3:0]) - This field records the type of request that generated the parity error as follows:

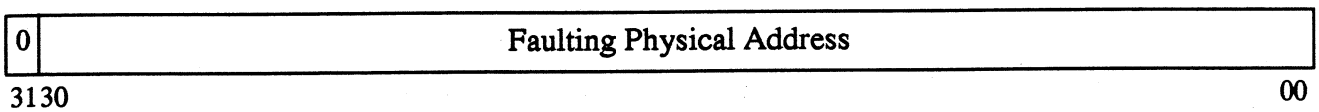
Table 32 - Memory Request Type

Value(Hex)	Name	Definition
0	NOP	No memory operation
1	RD64	Read of 64 bits (2 words)
2	RD128	Read of 128 bits (4 words)
3	-	Reserved
4	RD256	Read of 256 bits (8 words)
5-8	-	Reserved
9	WR8	Write of 8 bits (1 byte)
A	WR16	Write of 16 bits (2 bytes)
B	WR32	Write of 32 bits (1 word)
C	WR64	Write of 64 bits (2 words)
D-F	-	Reserved

5.7.9 Memory Fault Address Register

The Memory Fault Address Register (MFAR) records the 31 bit physical address that caused the fault. This register is accessed using SBus and IO MMU Control Space (0x10001054). This register is loaded on every request to memory unless it is locked. A hardware lock is used to ensure that this register does not change while being read if there was an error condition. Reading this register allows it to begin loading once again. Bit [31] should be written as zero and will be read as zero. The structure of this register is as follows.

Table 33 - Memory Fault Address Register

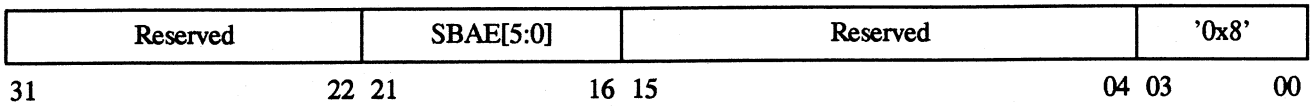


Note that bit 31 is unimplemented, should be written as zero, and will be read as zero. Also, this register is only held when an error is reflected in the MFSR.

5.7.10 MID Register

The MID Register contains two fields. The MID field (Bits[3:0] contain a constant value of 0x8) and the SBAE field which controls the ability of SBus devices to arbitrate for the bus. This register can be accessed using SBus and IO MMU Control Space (0x10002000).The SBAE bits are both readable and writeable while the MID field is read only. The MID is defined as follows:

Figure 5.23 - MID Register



Field definitions:

Reserved - Bits [31:21,15:04] are not implemented, should be written as zero, and will be read as zero.

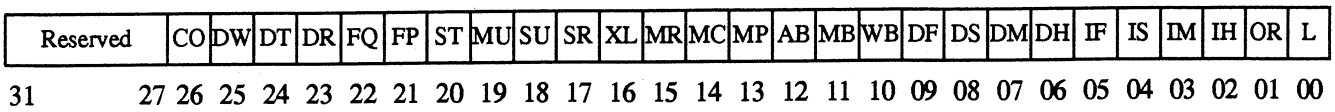
SBus Arbitration Enable[5:0] (SBAE) - These bits control the ability for devices on the SBus to arbitrate for the bus. The most significant bit (SBAE[5]) controls arbitration for the SCSI/ Ethernet master. The other bits (SBAE[4:0]) control arbitration for SBus devices 4:0 corresponding to SSCR[4:0]. These bits are R/W.

MID - This field is a constant 0x8 and is read only (writes to these bits are ignored).

5.7.11 Trigger A Enables Register

The Trigger A Enable Register is used to define trigger events for Performance Counter A. Setting a field to "1" will enable that trigger event for counting. This register can be accessed using SBus and IO MMU Control Space (0x10003000).

Figure 5.24 - Trigger A Enables Register



Field definitions:

Reserved - Bits [31:27] are not implemented, should be written as zero, and will be read as zero.

L - Trigger on edge of Level. When set to a "1" this bit will cause Triggers defined in this register to be level sensitive. When cleared to a "0" the trigger becomes edge sensitive.

OR - Combine triggers by ORing or ANDing function. When set to a "1", this bit enables the Logical OR of the specified triggers to be counted. If the bit is set to a "0", the triggers will be logically ANDed to form the increment signal.

IH - I-cache miss -Asserted 1-cycle after the miss is detected, and sustained until the miss has been resolved. (ssparc.iwait_f)

IM - I-cache miss pending - Asserted 1-cycle after miss detected, and sustained until corresponding memory request has been made. (ssparc.ic_miss)

IS - I-cache streaming - Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed. (ssparc.ssparc_mmu.MMU_cntl.ic_stream)

IF - I-cache lookup -

DH - D-cache miss -Asserted 1-cycle after the miss is detected, and sustained until the miss has been resolved. (ssparc.dwait_w)

DM - D-cache miss pending - Asserted 1-cycle after miss detected, and sustained until corresponding memory request has been made. (ssparc.dc_miss)

DS - D-cache streaming - Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed. (ssparc.ssparc_mmu.MMU_cntl.dc_stream)

DF - D-cache lookup

WB - Write buffer full - Asserted while all 4 write buffer entries are valid. (ssparc.dc_shold)

XL - Translation - 1-cycle pulse for each translation attempt. (ssparc.ssparc_mmu.MMU_cntl.r_tlb_used)

SR - Processor Tablewalk - Asserted for the duration of processor tablewalks. Can be used in conjunction with the translation count to determine TLB hit rate. (ssparc.ssparc_mmu.MMU_cntl.sr_tw)

SU - Supervisor mode - Based on the processor PSR.S bit. Can be used with other fields to determine supervisor overhead.

MU - MMU breakpoint - Combined signal from MMU breakpoint decode.

ST - Pipeline stalled - Asserted whenever the pipeline is stalled (1-cycle delay). (ssparc.iu_pipe_hold)

Memory request - Asserted once for each memory access. (ssparc.mm_issue_req)

MB - Memory busy- Memory currently busy.

AB - Local Graphics busy- Local Graphics interface currently busy.

MR - Memory RMW op- Memory Read/Modify/Write operation requested.

MP - Memory page mode access - Asserted once for each memory access that is on the same page as the previous access (for a given DRAM bank). (ssparc.mm_page)

MC - Memory precharge request - Asserted once for each memory access that indicated non-page hit prior to request. (ssparc.mm_precharge)

DR - DMA request -Asserted when DMA request is pending in the MMU. This is for both SBus translations and memory accesses.

DT - DMA tablewalk -I/O tablewalk signal is active 1-cycle after the translation miss, until complete.

DW - DMA stores -Active for DMA stores. Sustained until data is transferred to memory. DMA stores sub-word / sub-dw - Active once per request. These are read-modify-write memory operations.

DMA translate -Active from sb_ioreq -> issue_req/xlate

FP - fhold_perf - fpu hold signal asserted for fld/fst dependency cases, or fp queue is full and another FPop is in the pipeline. Guaranteed to hold the iu pipeline if psr.ef==1

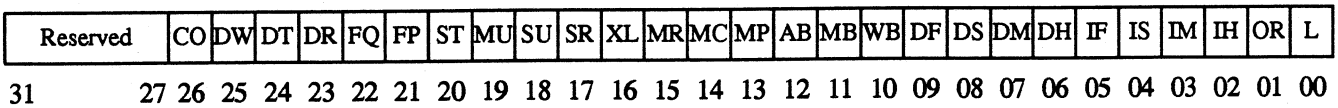
FQ - fhold_fq_full - indicates that fhold is asserted because the fp queue is full, and another FPop is in the pipeline.

CO - Counter B_CO -Counter B Carry out. For trigger A only, this allows Counter A to reflect the number of counter B overflows.

5.7.12 Trigger B Enables Register

The Trigger B Enable Register is used to define trigger events for Performance Counter B. Setting a field to "1" will enable that trigger event for counting. This register can be accessed using SBus and IO MMU Control Space (0x10003004).

Figure 5.25 - Trigger B Enables Register



Field definitions:

Reserved - Bits [31:27] are not implemented, should be written as zero, and will be read as zero.

L - Trigger on edge of Level. When set to a "1" this bit will cause Triggers defined in this register to be level sensitive. When cleared to a "0" the trigger becomes edge sensitive.

OR - Combine triggers by ORing or ANDing function. When set to a "1", this bit enables the Logical OR of the specified triggers to be counted. If the bit is set to a "0", the triggers will be logically anded to form the increment signal.

IH - I-cache miss -Asserted 1-cycle after the miss is detected, and sustained until the miss has been resolved. (ssparc.iwait_f)

IM - I-cache miss pending - Asserted 1-cycle after miss detected, and sustained until corresponding memory request has been made. (ssparc.ic_miss)

IS - I-cache streaming - Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed. (ssparc.ssparc_mmu.MMU_cntl.ic_stream)

IF - I-cache lookup -

DH - D-cache miss -Asserted 1-cycle after the miss is detected, and sustained until the miss has been resolved. (ssparc.dwait_w)

DM - D-cache miss pending - Asserted 1-cycle after miss detected, and sustained until corresponding memory request has been made. (ssparc.dc_miss)

DS - D-cache streaming - Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed. (ssparc.ssparc_mmu.MMU_cntl.dc_stream)

DF - D-cache lookup

WB - Write buffer full - Asserted while all 4 write buffer entries are valid. (ssparc.dc_shold)

XL - Translation - 1-cycle pulse for each translation attempt. (ssparc.ssparc_mmu.MMU_cntl.r_tlb_used)

SR - Processor Tablewalk - Asserted for the duration of processor tablewalks. Can be used in conjunction with the translation count to determine TLB hit rate. (ssparc.ssparc_mmu.MMU_cntl.sr_tw)

SU - Supervisor mode - Based on the processor PSR.S bit. Can be used with other fields to determine supervisor overhead.

MU - MMU breakpoint - Combined signal from MMU breakpoint decode.

ST - Pipeline stalled - Asserted whenever the pipeline is stalled (1-cycle delay). (ssparc.iu_pipe_hold)

Memory request - Asserted once for each memory access. (ssparc.mm_issue_req)

MB - Memory busy- Memory currently busy.

AB - Local Graphics busy- Local Graphics interface currently busy.

MR - Memory RMW op- Memory Read/Modify/Write operation requested.

MP - Memory page mode access - Asserted once for each memory access that is on the same page as the previous access (for a given DRAM bank). (ssparc.mm_page)

MC - Memory precharge request - Asserted once for each memory access that indicated non-page hit prior to request. (ssparc.mm_precharge)

DR - DMA request -Asserted when DMA request is pending in the MMU. This is for both SBus translations and memory accesses.

DT - DMA tablewalk -I/O tablewalk signal is active 1-cycle after the translation miss, until complete.

DW - DMA stores -Active for DMA stores. Sustained until data is transferred to memory. DMA stores sub-word / sub-dw - Active once per request. These are read-modify-write memory operations.

DMA translate -Active from sb_ioreq -> issue_req/xlate

FP - fhold_perf - fpu hold signal asserted for fld/fst dependency cases, or fp queue is full and another FPop is in the pipeline. Guaranteed to hold the iu pipeline if psr.ef==1

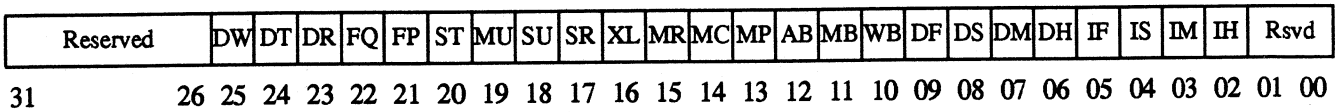
FQ - fhold_fq_full - indicates that fhold is asserted because the fp queue is full, and another FPop is in the pipeline.

CY - Cycle count - Always active.

5.7.13 Assertion Control Register

The Assertion Control Register can be used to "invert" any trigger event defined in the two trigger Registers. Setting a field to "1" will cause the trigger event for that field to be inverted prior to going into the trigger register logic. This register can be accessed using SBus and IO MMU Control Space (0x10003008).

Figure 5.26 - Assertion Control Register



Field definitions:

Reserved - Bits [31:26,01:00] are not implemented, should be written as zero, and will be read as zero.

IH - I-cache miss -Asserted 1-cycle after the miss is detected, and sustained until the miss has been resolved. (~ssparc.iwait_f)

IM - I-cache miss pending - Asserted 1-cycle after miss detected, and sustained until corresponding memory request has been made. (~ssparc.ic_miss)

IS - I-cache streaming - Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed. (~ssparc.ssparc_mmu.MMU_cntl.ic_stream)

IF - I-cache lookup -

DH - D-cache miss -Asserted 1-cycle after the miss is detected, and sustained until the miss has been resolved. (~ssparc.dwait_w)

DM - D-cache miss pending - Asserted 1-cycle after miss detected, and sustained until corresponding memory request has been made. (~ssparc.dc_miss)

DS - D-cache streaming - Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed. (~ssparc.ssparc_mmu.MMU_cntl.dc_stream)

DF - D-cache lookup

WB - Write buffer full inverted. (~ssparc.dc_shold)

XL - Translation inverted. (~ssparc.ssparc_mmu.MMU_cntl.r_tlb_used)

SR - Processor Tablewalk inverted. (~ssparc.ssparc_mmu.MMU_cntl.sr_tw)

SU - Supervisor mode inverted.

MU - MMU breakpoint inverted.

ST - Pipeline stalled inverted). (~ssparc.iu_pipe_hold)

Memory request inverted. (~ssparc.mm_issue_req)

MB - Memory busy inverted.

AB - Local Graphics busy inverted.

MR - Memory RMW op inverted.

MP - Memory page mode access inverted. (~ssparc.mm_page)

MC - Memory precharge request inverted. (~ssparc.mm_precharge)

DR - DMA request inverted.

DT - DMA tablewalk inverted.

DW - DMA stores inverted.

DMA translate inverted

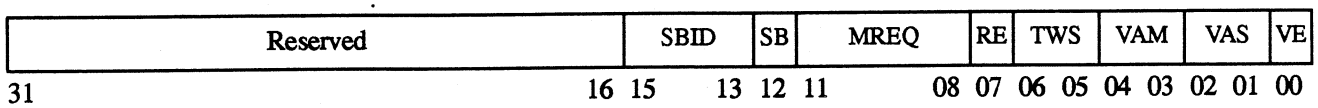
FP - FPU pipeline hold inverted.

FQ -FPU Queue full inverted.

5.7.14 MMU Breakpoint Register

The MMU Breakpoint Register can specify a single breakpoint based on a number of field comparisons defined in the register. Any of the fields can be used as a stand alone compare, or combined with other fields that are part of the MMU function. This register can be accessed using SBus and IO MMU Control Space (0x1000300C)

Figure 5.27 - MMU Breakpoint Register



Field definitions:

Reserved - Bits [31:11] are not implemented, should be written as zero, and will be read as zero.

VE - Virtual Address Breakpoint enable.

VAS - Virtual Address Source. These bits are used to control the source of the address to be used for breakpoint compare operations. The three bits are defined in the following table:

Table 34 - MMU Breakpoint Register VAS Field decode

VAS	Virtual Address Source
00	I-Cache Address
01	D-Cache Address (includes write buffer)
10	DVMA Address(for I/O MMU functions)
11	Physical Address

VMA - Virtual Address Memory operation. These bits are used to describe the type of memory operation to be used together with the address compare for breakpoint. The possible memory ops for breakpoint definition are shown in the following table::

Table 35 - MMU Breakpoint Register VAM Field decode

VAM	Virtual Address Memory operation
00	disabled
01	read (D-Cache miss, I-Cache miss, or DVMA)
10	write (D-Cache miss or DVMA)
11	LDSTO or DVMA translate

TWS - Table Walk translation Source. These bits are used to describe the type of table walk operation to be used for the breakpoint. The bit definitions are shown in the following table::

Table 36 - MMU Breakpoint Register TWS Field decode

TWS	TableWalk translation Source
00	disabled
01	Instruction Fetch Table Walk
10	Data op Table Walk
11	DVMA Table Walk

RE - Memory Request compare Enable. This bit field is used to enable breakpoint compare on the size and type information for memory operations. When used with the MT field this enable allows breakpoints on operations of a specific size to be enable.

MT - Memory request Type. This bit field is used to define the type and size of the memory operation for the breakpoint event. The

possible definitions for this field are shown in the following table:

Table 37 - MMU Breakpoint Register MT Field decode

MT	Memory Request Type
0000	nop
0001	READ 8 bytes
0010	READ 16 bytes
0011	Reserved
0100	READ 32 bytes
0101 - 1000	Reserved
1001	WRITE 1 byte
1010	WRITE 2 bytes
1011	WRITE 4 bytes
1100	WRITE 8 bytes
1101	WRITE 16 bytes
1110 - 1111	Reserved

SB - SBus slot ID match enable. When set to a "1" this bit enables the use of SBus slot ID match to be used in the breakpoint logic. When cleared to a "0", the SBus ID match logic does not affect breakpoint detection.

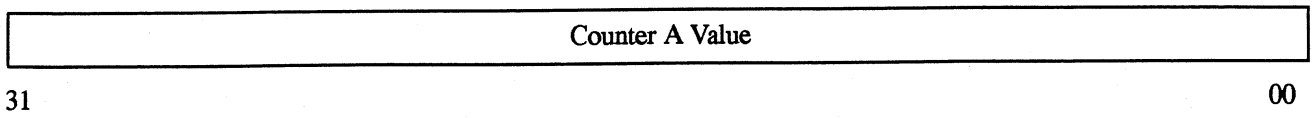
SBID - SBus Slot ID match target. This field is used to define the target SBus Slot ID to use for breakpoint match operations when the SB bit has been set.

5.7.15 Performance Counter A

Performance Counter A is the first of two 32 bit counters. Counter A will be incremented based on the assertion of triggers defined for

counter A in the Trigger A Enables Register. Performance Counter A can be accessed using SBus and IO MMU Control Space (0x10003010)

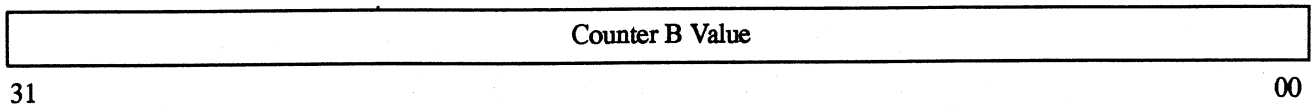
Figure 5.28 - Performance Counter A



5.7.16 Performance Counter B

Performance Counter B is the second 32 bit counter. Counter B will be incremented based on the assertion of triggers defined for counter B in the Trigger B Enables Register. Performance Counter B can be accessed using SBus and IO MMU Control Space (0x10003014).

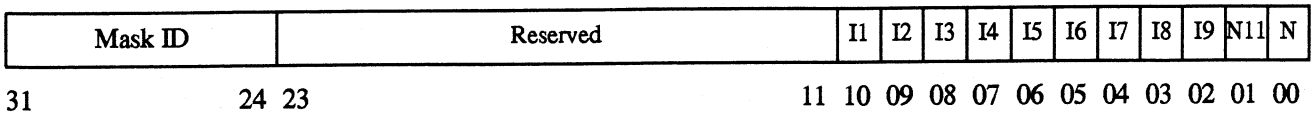
Figure 5.29 - Performance Counter B



5.7.17 VirtualAddress Mask Register

The Virtual Address Mask Register is used to disable the compare of specific bit fields in the Virtual Address Compare Register. Enables I1 - I9 enable their respective fields for comparison. The N11 and N bits are used to decode the 'compare not' function. The N11 bit only affects the F field (VA[11]), and the N bit affects the range of VA[31:12]. When the N=0, normal comparisons are made. When N=1, the compare result is inverted - so a 'hit' occurs when the addresses mismatch. This register can be accessed using SBus and IO MMU Control Space (0x10003018).

Figure 5.30 - Virtual Address Mask Register



Field definitions:

Reserved - Bits [31:11] are not implemented, should be written as zero, and will be read as zero.

N - Normal or inverted comparison enable. N=0 enables normal comparisons, N=1 enables inverted comparisons.

N11 - Normal of inverted compare mode for bit 11 only.

I1 - Compare enable for VA[31:24].

I2 - Compare enable for VA[23:18].

I3 - Compare enable for VA[17:12].

I4 - Compare enable for VA[11].

I5 - Compare enable for VA[10:04].

I6 - Compare enable for VA[03].

I7 - Compare enable for VA[02].

I8 - Compare enable for VA[01].

I9 - Compare enable for VA[00].

Mask ID - This read-only eight bit field is used to uniquely identify the revision level of the mask that was used to manufacture the part. The revision number shall be one of the entries found in the following table:

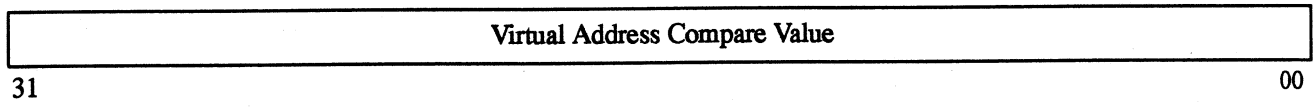
Table 38 - TLB Entry Address Mapping

Mask ID	Mask Revision
0000 0000	1.0 First Tapeout
0001 0001	1.1 Metal Only Tapeout
:0010 0000	2.0 Second Tapeout
0010 0100	2.4 Metal Only Tapeout
0011 0000	3.0 Third Tapeout

5.7.18 Virtual Address Compare Register

The Virtual Address Compare Register is used to set the value of the Virtual address that the breakpoint logic will use to compare against. This register shold be used together with the Virtual Address Mask Register to define the exact match criteria for the breakpoint. The VA can be either the I-cache, D-cache or DMA virtual address, or the address that is being translated by the MMU. Since the caches are virtually tagged, cache hit accesses do not need to be translated. Therefore, selects are provide to maintain address checking on a single source of address (regardless of hit/miss results). This register can be accessed using SBus and IO MMU Control Space (0x1000301C).

Virtual Address Compare Register

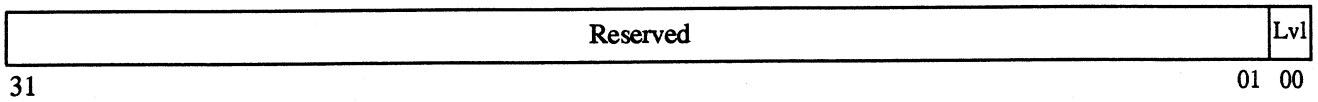


5.7.19 Local Graphics Queue Level Register

The Local Graphics Queue Level Register is used to set the value of the threshold for CPU Local Graphics read "hold off". If the number of Local Graphics operations, in the Local Graphics queue is greater than the set threshold, CPU Local Graphics operations will not be issued. The

CPU will be held until such time as the Local Graphics queue is at a level that would allow the operation to be issued. During this hold time DVMA may freely access main memory. The threshold may only be set to 0x0, or 0x1. Note that this register is accessed via two different addresses. For writing this register, writes must be done to address 0x10004000. Writes to 0x10006000 will NOT update the register contents. For reading this register, reads must be done to address 0x10006000, reads to 0x10004000 will NOT return the current register contents. All reserved bits hsould be written as "0", and shall be read as "0". This register can be accessed using SBus and IO MMU Control Space (0x10004000 for writes, and 0x10006000 for reads).

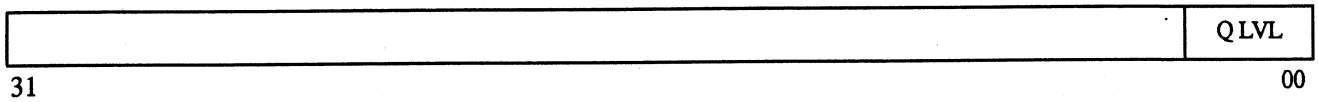
Figure 5.31 - Local GraphicsQueue Level Register



5.7.20 Local Graphics Queue Status Register

The Local Graphics Queue Status Register is a READ ONLY register that reflects the current number of Local Graphics operations in the Local Graphics queue. When read, this register should have a value of between 0x0 and 0x4. All reserved bits shall be read as "0".This register can be accessed using SBus and IO MMU Control Space (0x10007000).

Figure 5.32 - Local GraphicsQueue Status Register



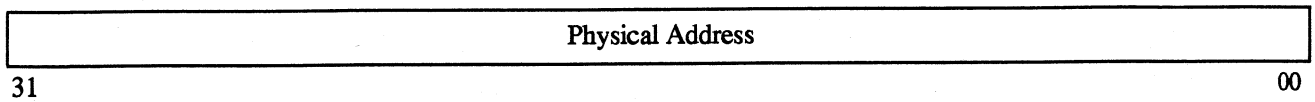
5.8 IO MMU Bypass Mode

Bypass mode is provided to allow intelligent SBus masters to do their own memory management with assistance from the kernel. This facility is enabled by having the Bypass Enable bit set in that device's slot configuration register. It is assumed that such a master will have its own MMU. In order to bypass the IO MMU the DVMA master must issue a virtual address with sb_ioa[31:30]=0. In this case the Physical Address bus will have the Virtual Address bus put on it. The PA is checked to verify that it is in the valid main memory range and an error is issued to the master if it is not.

5.9 Physical Address Register

The Physical Address Register (PAR) is used to hold translated physical addresses before they are used for either memory requests or for SBus operations. This register cannot directly be read or written. The structure of this register is as follows.

Figure 5.33 - Physical Address Register



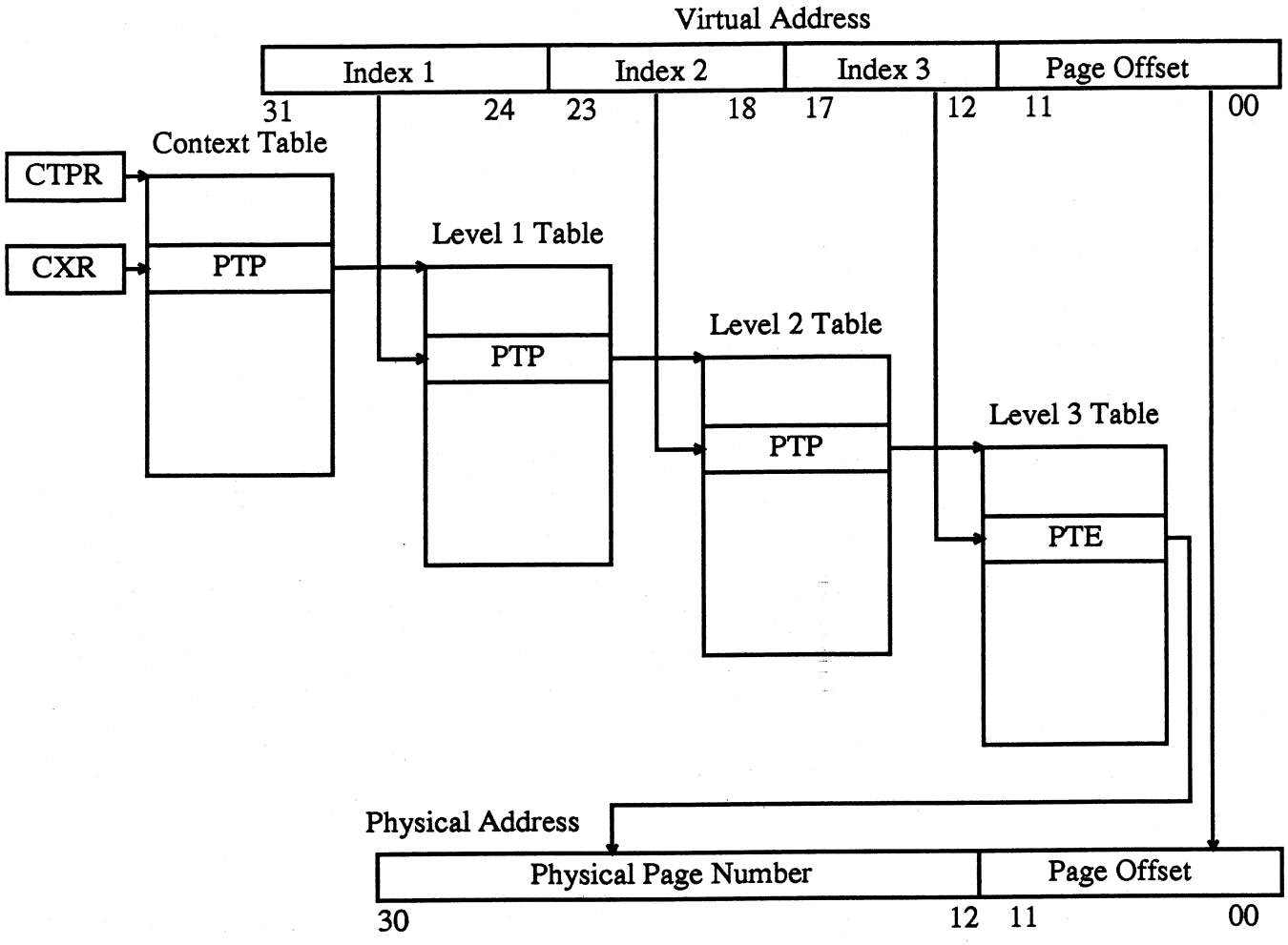
In order to avoid redundant retranslation, two extra registers are used to store PAR data. They are DPAR(Data Physical address register), and IPAR(Instruction Physical address register).

5.10 TLB Table Walk

On a translation miss the table walk hardware translates the virtual address to a physical address by "walking" through a context table and from 1 to 3 levels of page tables. The first and second levels of these tables typically (not necessarily) contain page table pointers (PTP) to the next level tables when accesses are due to CPU instruction or data addresses. IO accesses only the first level page table. A third level table entry should always be a page table entry (PTE) pointing to a physical page or else a translation fault occurs.

The table walk for a CPU generated virtual address uses the context table pointer register (CTPR) as a base register and the context number contained in the context register (CXR) as an offset to point to an entry in the context table. The context table entry is then used as a PTP into the first level page table. At any address the table walk hardware finds either a PTE which terminates its search or a PTP. A PTP is used in conjunction with a field in the virtual address to select an entry in the next level of tables. The table walk continues searching through levels of tables as long as PTPs are found pointing to the next table. The table walk terminates when either a PTE is found or an exception is generated if a PTE is not found after accessing the 3rd level page table (or if an invalid or reserved entry is found). Note that PTPs and PTEs encountered during a table walk are not cached in the data cache. A full table walk is shown in the following figure.

Figure 5.34 - CPU Address Translation Using Table Walk



When the PTE is found it is stored in a TLB entry, indexed by the TLBRC, and used to complete the original virtual to physical address translation. A table walk which was forced by a store operation to an unmodified region of memory causes the M bit in the PTE to be set. Any "entire" probe or normal tablewalk operation causes the R bit of the PTE to be set if it had not been already.

This table walk is changed when virtually tagged level 2 PTPs are enabled. In the case of virtually tagged the MMU will initially search for the level 2 PTP by using the CXR, Index 1, and Index 2 of the virtual address. Should this PTP be found in the TLB, there is no need for the Context table and level 1 lookups. We may then use the level 2 PTP to lookup the required PTE directly. This effectively cuts the table walk in

half. Should the virtual tagged PTP not be found in the TLB, the MMU will start the table walk with the context table.

The table walk for an IO generated virtual address uses the IO Base Address Register (IOBAR) as a base register and part of the DVMA virtual address as an index into an IOPTE table in memory. Specifically the IO MMU page table size and corresponding DVMA virtual address range are configured in the IOCR RANGE field. The table consists of 4 byte entries. The virtual address used for this mapping is VA[X:0] where "X" is the highest VA bit in the translatable range. VA[31:X+1] must be all "1"s in order for translation to take place; otherwise an error is signalled to the DVMA master. The bits VA[X:12] provide a virtual page number which is used as an index into the IOMMU table in memory. These bits are placed on mm_pa[X-10:2]. The rest of the physical address is mm_pa[1:0] = 00, and mm_pa[30:X-9] = IBA[30:X-9]. This is the PA used for the one level IO walk.

5.11 Arbitration

The MMU block performs the primary memory arbitration function on the CPU. This is due to the central nature of the MMU in the address flow of the machine. The different sources of memory activity are the instruction cache block (for instruction fetches), the data cache block (for loads and stores), the TLB (during tablewalks and to keep the referenced and modified bits in the main memory page tables up to date), and IO DMA activity.

The other entity needing main memory is the DRAM refresh logic. This function is folded into the arbitration scheme by the Memory Controller which must arbitrate between it and a request out of the MMU.

The arbitrating requirements can be broken down into several different resource arbiters. The TLB (and ITLB) arbitration and the internal memory bus arbitration.

5.11.1 TLB Arbitration

The current priority scheme places TLB references as highest priority, followed by IO references, data references, and finally instruction references. Tablewalks and updates to the memory PTEs due to changes

to the Referenced and Modified bits are the highest priority. They imply that some other operation is in progress.

Table 39 - TLB Reference Priority

Operation pending			Results
IO DMAs	IU Data Ref	Instr. Fetch	
Yes	X	X	Xlate for IO, Tablewalk if miss
No	Yes	X	Xlate for IU Data Reference, Table-walk if miss
No	No	Yes	Xlate for Instruction Fetch, Table-walk if miss

Note: X=Don't Care, Xlate=Translate

5.12 Translation Modes

Translation of virtual addresses to physical addresses is done in the following modes:

Table 40 - Translation Modes

Name	ASI	Boot Mode	MMU En.	PA[30:00]
Boot IFetch	0x8, 0x9	Yes	X	PA[30:28]=0x7, PA[27:00]=VA[27:00]
Pass Through	0x8, 0x9	No	Off	PA[30:00]=VA[30:00]
Translate	0x8, 0x9	No	On	PA[30:12]=PTE[26:08], PA[11:00]=VA[11:00]
Pass Through	0xA, 0xB	X	Off	PA[30:00]=VA[30:00]
Translate	0xA, 0xB	X	On	PA[30:12]=PTE[26:08], PA[11:00]=VA[11:00]
Bypass	0x20	X	X	PA[30:00]=VA[30:00]

5.12.1 Page Hit Registers

The MMU is responsible for generating a signal to the memory controller indicating whether or not the current memory request can use page mode of the DRAMs or not. This is done by comparing the current physical address (mm_pa) against the physical addresses of the previous memory access to the even and odd memory banks. For this purpose the

MMU has two registers which are used to store the current physical address. Register 0 is used if the memory operation is to an "even" bank. Register 1 is used if the memory operation is to an "odd" bank. The even or odd state of an address is determined by bit 25 of the physical address. For the page hit registers a "bank" refers to the physical address space consumed by a single DRAM sim module. Each register is used to store bits 30:12 of the physical address. If either register detects that the current access is to the same bank as the previous access the page mode signal to the memory controller will be acitivated. This signal can be over-ridden by the control bits in the PCR register to disable this feature.

5.13 Errors and Exceptions

The MMU generates: instruction access error, instruction access exception, data access error, and data access exception for the SPARC IU. Also, an external interrupt is driven for asynchronous faults. In a Sun4M system, this would indicate a level 15 interrupt.

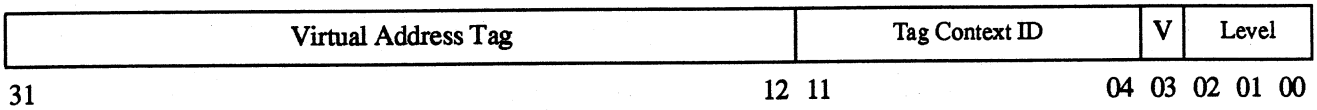
5.14 Diagnostic Features

All registers and RAM (and CAM) are accessible directly through alternate virtual address space loads and stores. There is also the ability to breakpoint on certain conditions. This is set up through use of the various Breakpoint/Performance counter registers.

5.14.1 Diagnostic Access of TLB

Diagnostic reads and writes to the 64 TLB entries are performed by using load and store alternate instructions in ASI 0x6 and the virtual address to explicitly select a particular TLB entry. The access must be a word access, all other data sizes will result in an internal error. Depending on the virtual address specified either the TLB Tag, or TLB PTE will be referenced. The format for the TLB PTE is as described earlier. The format of the Tag is shown below:

Figure 5.35 - CPU Diagnostic TLB Upper Tag Access Format



Field Definitions:

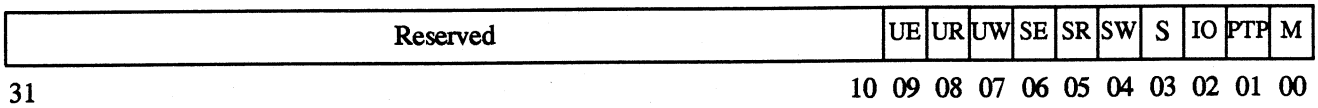
Virtual Address Tag - The 20 bit virtual address tag represents the most significant 20 bits (VA[31:12] the page address) of the

virtual address being used. VA[11:00] is the byte within a page. The address in this field is physical when referencing physically tagged PTPs with the least significant 20 bits containing PA[27:08].

Context Tag - The 8 bit context tag comes from the value in the context register as written by memory management software. Both it and the virtual address tag must match the CXR and VA[31:12] in order to have a TLB hit. This field contains a physical address {(PA[07:02]) + 2'b00}when referencing PTPs.

Valid bit, Level bits - These 3 bits are used to enable the proper virtual tag match of root, region, and segment PTE's. The Valid bit indicates a valid entry. Bit 2 is the index bit for the root, bit 1 is the index for the region, and bit 0 is the index for the segment.

Figure 5.36 - CPU Diagnostic TLB Lower Tag Access Format



Field Definitions:

Unused - These bits are not used and will always return zero.

UE - This bit indicates that the Page has User Execute permission set.

UR - This bit indicates that the Page has User Read permission set.

UW - This bit indicates that the Page has User Write permission set.

SE - This bit indicates that the Page has Supervisor Execute permission set.

SE - This bit indicates that the Page has Supervisor Read permission set.

SW - This bit indicates that the Page has Supervisor Write permission set.

Modified (M) - This bit is used to indicate that the page has been modified by a previous write operation.

IO Page Table Entry (IO) - This bit indicates that an IOPTE resides in this entry of the TLB. This bit is non meaningful for an ITLB Tag and is read as 0.

Page Table Pointer (PTP) - This bit indicates that a PTP resides in this entry of the TLB. Note that all SRMMU flush types (except page) will flush all PTPs from the TLB. This bit is non meaningful for an ITLB Tag and is read as 0.

Note that when loading TLB entries under software control (using alternate space accesses) care should be taken to ensure that multiple TLB entries cannot map to the same virtual address. This may inadvertently occur when combining TLB entries that map different sizes of addressing regions. A level 3 PTE could be included in a TLB region for a level 1 or 2 PTE for example. The TLB output is not valid when this occurs.

The virtual address is used to select the TLB entries as follows:

Table 41 - TLB Entry Address Mapping

Virtual Address	TLB Entry
0x0	Entry 0 PTE
0x4	Entry 1 PTE
:	:
0xF8	Entry 62 PTE
0xFC	Entry 63 PTE
0x100	Entry 0 Tag[9:0]
0x104	Entry 1 Tag[9:0]
:	:
0x1F8	Entry 62 Tag[9:0]
0x1FC	Entry 63 Tag[9:0]
0x200-0x2FF	Reserved
0x300	Entry 0 Tag[41:10]
0x304	Entry 1 Tag[41:10]
:	:
0x3F8	Entry 62 Tag[41:10]
0x3FC	Entry 63 Tag[41:10]
0x400-FFFFFFFC	Reserved

5.14.2 MMU Breakpoint Debug Logic

The MMU breakpoint debug logic is intended for use in lab debug only since it requires setup through a scan facility. The basic idea is to stop the clocks when certain conditions occur. This facility is general purpose in that there is a large matrixed selection of conditions to choose from. The breakpoints which can be enabled are virtual address matching, virtual address source matching (includes type of request), memory request matching, tablewalk detection (includes type), tablewalk level matching, and SBus slot ID matching. A more detailed description and suggested pairings of these conditions follows.

We have the ability to breakpoint on portions of the virtual address (the output of the virtual address muxing logic). These portions of the virtual address can be combined with other conditions to make their match conditions more case specific as follows:

Table 42 - Virtual Address Match Conditions

Virtual Address Conditions	Conditions to be Paired With
VA[31:00] VA[31:01] VA[31:02] VA[31:03] VA[31:12] VA[31:18] VA[31:24] VA[10:02] VA[11:02] !VA[31:12] & VA[11:02] !VA[31:11] & VA[10:02]	ic_tlb & iu_fetch_f (valid instruction fetch - D stage) dc_tlb & read_w (valid data read - W stage) dc_tlb & write_w (valid data write - W stage) dc_tlb & ldsto_w (valid data atomic op - W stage) io_tlb & sb_read (valid DMA read) io_tlb & sb_write (valid DMA write - Physical address) io_tlb & sb_translate (valid DMA translate - before write)

We also have the ability to breakpoint on the particular type of memory request being sent from the MMU to the MEMIF. This is sampled when a memory request is actually being issued (mm_issue_req = 1). This can be paired with two other fields indicating the type of tablewalk

occurring and the tablewalk level to match (if memory request indicates a tablewalk) as follows:

Table 43 - Memory Request Type

Memory Request	
NOP	No memory operation
RD64	Read of 64 bits (2 words)
RD128	Read of 128 bits (4 words)
RD256	Read of 256 bits (8 words)
WR8	Write of 8 bits (1 byte)
WR16	Write of 16 bits (2 bytes)
WR32	Write of 32 bits (1 word)
WR64	Write of 64 bits (2 words)
Tablewalk Type	
None	No tablewalk in progress
ic_tlb_tw	Tablewalk from instruction fetch
dc_tlb_tw	Tablewalk from data reference
io_tlb_tw	Tablewalk from DVMA
Tablewalk Level	
Root Level	
Level 1	
Level 2	
Level 3	

We also have the ability to breakpoint on the particular 3 bit SBus slot ID. This event can be paired with the type of io request or a virtual address match to make it more specific.

5.14.3 Additional Features

There are other features which can be used for microSPARC-II debug. Some of these features are enabled using Processor Control Register bits. Software tablewalks can be enabled by asserting PCR[23], the STW bit. When in this mode the MMU will cause the instruction_access_MMU_miss and data_access_MMU_miss traps for instruction and data tablewalking respectively for tablewalks to be done by software.

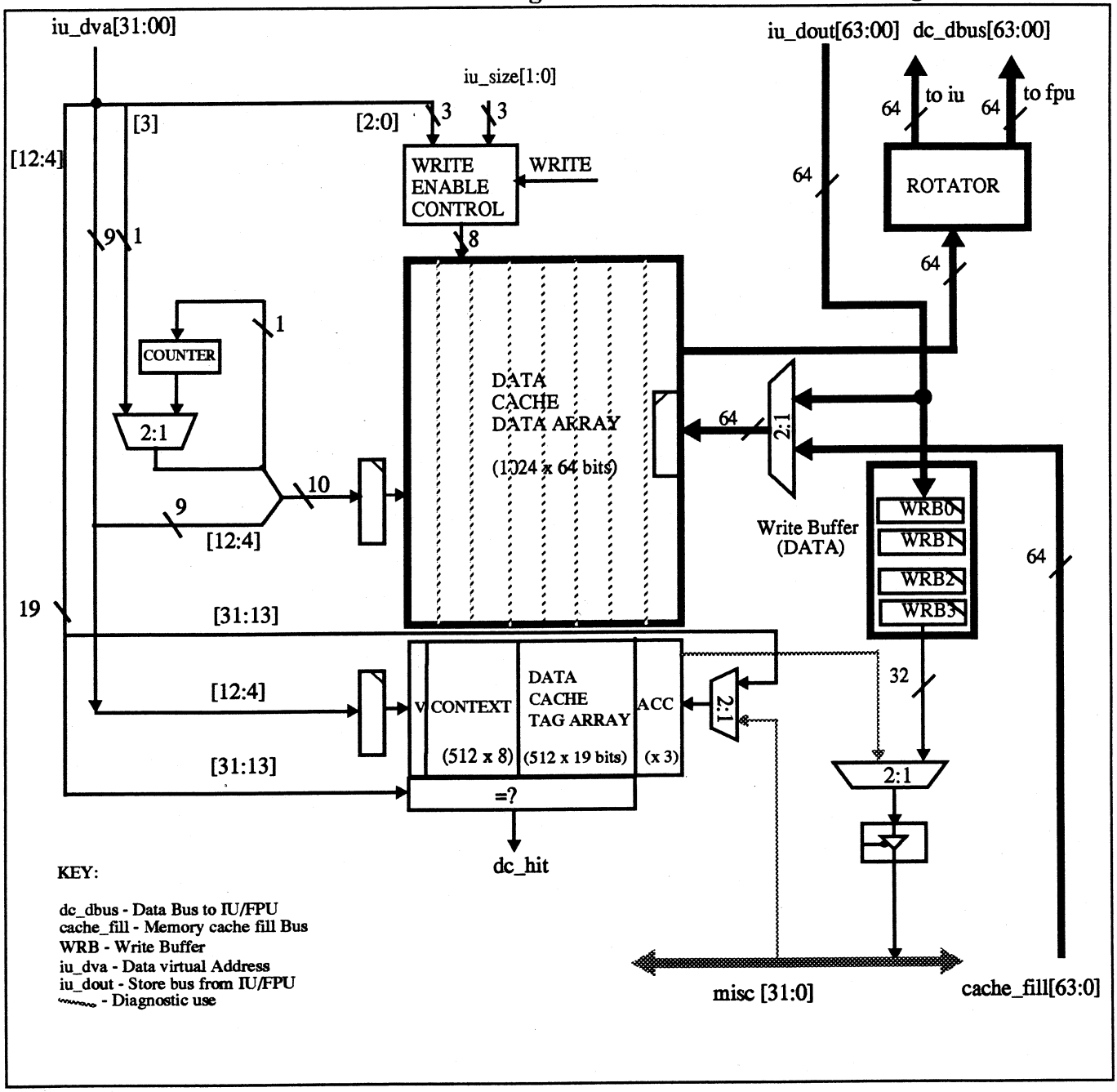
Chapter 6 Data Cache

6.1 Overview

The microSPARC-II Data Cache is a 8K-Byte, direct mapped Cache, used on Load or Store accesses from the CPU to cacheable pages of main memory. It is virtually indexed and virtually tagged. Stores are write-through with write allocate. The Data Cache is addressed by iu_dva[12:0]. The Data Cache is organized as 512 lines of 16 bytes of data. Each line has a cache tag store entry associated with it. There is no sub-blocking. On a Data Cache miss to a cacheable location, 16 bytes of data are written into the cache from main memory. The cache tags contain a 19 bit DVA[31:13] tag field, an 8 bit context field, 3 bit ACC field, and 1 valid bit.

Within the Data Cache block there are cache bypass paths. These paths are used for noncached load references, and for streaming data into the IU or FPU on cache misses. A simple block diagram follows.

Figure 6.0 - Data Cache Block Diagram



6.2 Data Cache Data Array

All IU write operations to cached locations write the data through to main memory, i.e. on a write hit, both the Data Cache and main memory are updated. On cache misses, the missed cache line is read from memory into the cache, i.e. write allocate. This was a way to avoid the stale data problem in the virtual Data Cache due to aliasing. It was

chosen against a write invalidate policy because it makes the Data Cache controller design more uniform, since load misses are also handled in a similar way.

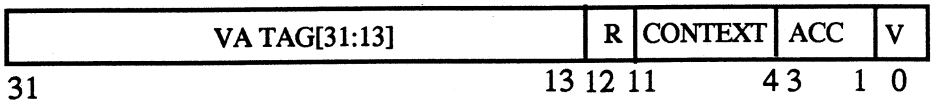
Diagnostic software may read and write the Data Cache directly by executing load or store alternate space instructions, of **any size**, in ASI space 0xF. Virtual address bits [12:0] will be used to address the Data Cache in this mode (adresses roll-over to the proper cache line, modulo 512 cache lines); all other virtual address bits as well as the Context ID bits, ACC bits and the Valid bit are ignored during these operations.

There are two input sources to the Data Cache data array. The IU data_out bus (iu_dout) is used when the Data Cache is updated on an integer or floating-point store operation. The memory cache fill bus (cache_fill[63:0]) is used as input for fills on Data Cache misses, and also for diagnostic ASI [0xC, 0xD, 0xE] loads.

6.3 Data Cache Tags

A Data Cache tag entry consists of several fields as follows.

Figure 6.1 - Data Cache Tag Entry



Field Definitions:

Virtual Address Tag - This field contains the virtual address of the data held in the cache line. The Data Cache Controller writes this field from bits [31:13] of the virtual address (iu_dva) of the line.

ACC bits - These are 3 bits indicating various levels of protection for that cache line. This is copied from the TLB.(These bits act as a superset of the S/W bits in previous SUN systems).

CONTEXT bits - These indicate the context of the particular cache line. They are filled from the TLB.

Valid bit - This bit indicates that the line contains data. This bit is set when a cache line is filled due to a successful cache miss; a cache fill which results in a memory parity error will leave the Valid bit unset. Stores to asi space (0x10-0x14) will clear the valid bits, conditionally, as defined in the SPARC reference manual.

R bit - This bit is reserved (Not to be confused with the Reference bit in the TLB).

There are two input sources to the Data Cache tag array. The Virtual Address bits (DVA[31:13]) are used for cache tag updates on Data

Cache misses. The miscellaneous bus (misc[31:0]) is used for store operations to ASI space (0xC, 0xD, 0xE) and to empty the store buffer contents to main memory.

Diagnostic software can read and write the Data Cache tags by executing only word-length LDA and STA (Load and Store Alternate) instructions in ASI space 0xE. The Virtual address bits [12:4] will select one of the 512 tags; all other address bits are ignored.

6.4 Write Buffers

The Write Buffers(WRB) are four, 64-bit registers in the Data Cache block used to hold data being stored from the IU or FPU to memory or other physical devices. WRB holds the store data until it has been sent over the misc bus to the destination device. For halfword or byte stores, this data is left-shifted (with zero-fill) into proper byte alignment for writing to a word-addressed device (and the resulting word is replicated to make a 64bit doubleword), before being loaded into one of the WRB registers. There is **no** diagnostic read/write access to the WBR registers. The WRB is emptied during (actually before completion of) a LD/ST miss fill sequence to avoid any stale data from being read in to the Data Cache on a miss to a virtual alias. Thus no snoop logic is needed to check for any data hazards between the WRB and the data coming back from main memory. The address portion of the WRB contains virtual addresses, as opposed to Physical addresses. Thus the need for translation on store hits is avoided until the store is to be written to memory. The WRB is filled from the Data Cache controller and it is emptied by the MMU controller. There is an array of 4 valid bits associated with each entry of the WRB. On a Store which Traps, the WRB is properly flushed by the Data Cache controller, while the IU pipeline is held by the Data Cache controller.(This is needed, because the WRB is written at the end of the E pipeline stage, and the store could trap in the W stage of the pipeline).

6.5 Data Cache Fill

The cache line size fetched from memory on Data Cache misses is 16 bytes. Memory will always return 16 bytes of data starting with the requested doubleword first followed by the other doubleword, which will wrap around a 16 byte boundary until the entire 16-byte block has been returned. The transfer rate is one doubleword every 4 or 5 cycles from memory (one double word, then 3 or 4 dead cycles). The two values are for the two values of the'sp_sel' input pin of the microSPARC-II chip. The Cache array is written 1 clock cycle after each word appears on the cache_fill bus. The following table illustrates the fill operation showing the order that words are written into the cache:

Table 44 - Data Cache Fill Ordering

Requested Word	Order of fill (modulo 16B)
0	(0, 1), 3/4 dead cycles, (2, 3)
1	(0, 1), 3/4 dead cycles, (2, 3)
2	(2, 3), 3/4 dead cycles, (0, 1)
3	(2, 3), 3/4 dead cycles, (0, 1)

During the write cycles of a cache fill, data can be bypassed (or "streamed") to the IU or FPU, a cycle after it appears on the cache_fill bus. During the dead cycles data from **any line in the cache** can be written/read from by subsequent ld/st instructions.

The IU is held on a cache miss for both (Ld/ST) in the W cycle, the protections checked in the TLB and, for both LD's and ST's, the pipe is held atl east until the first requested word of the line is back.

6.6 ASI/STore Bus Interface

The Data Cache block interfaces to the misc bus for ASI ST/LD operations. Data from the Data Cache block to misc bus comes from the WRB. There are control signals from the MMU and Memory Controller to indicate when data is on cache_fill bus, to be loaded into the Data Cache and when data from the write buffer is to be driven onto the misc bus.

6.7 Cache fill Bus Interface

The Data Cache is filled by a separate 64bit bus (cache_fill bus), from main memory. Keeping this bus separate from the other buses on the chip, gives the flexibility to tune this bus for higher performance, without incurring a large chip area. By keeping this bus separate, it is possible to use a 64 bit cache fill, rather than a 32 bit fill. This bus can also be bypassed directly (after being latched inside the Data RAM) to the IU / FPU, through a mux in the data RAMs. This bypass path is used for streaming and for non-cached loads.

6.8 IU/FPU Data Bus Interface

The Data Cache block interfaces to an input and output IU/FPU data bus (iu_dout and dc_do). Data to the IU or FPU is sourced from either the latched output of the cache_fill bus (for streamed data on Data Cache misses, and for non-cached loads) or the Data Cache (for Data Cache hits). Load data to the IU/FPU has to go through a "rotator" block, which aligns the 64 bit word from memory or the cache to the IU/FPU. Data from the IU or FPU on store operations is loaded into the WRB and written into the cache RAMs. The interface to the FPU is 64b wide for both LD's and ST's, whereas the IU only has a 32b interface.

6.9 Data Cache Flushing

The Data Cache tags are implemented with all the five flush mechanisms (Page, Segment, Region, Context and User) as suggested in the reference MMU document. These are activated by word size store instructions to ASI 0x10 - ASI 0x14. The **addressed** Data Cache line's (addressed by iu_dva[12:04]) valid bit is reset (to zero) by this operation, if it matches. Note that the Data Cache is not flushed by the FLUSH instruction (the addressed instruction cache line however is). The store alternate flushes however flush **both** the Data Cache and the Instruction Cache, (although not neccessarily in exactly the same clock cycle). Another way to flush both the caches is by explicitly writing a 0x0 into the Valid bit of the cache line using the cache tag diagnostic ASIs (0xC for the instruction cache and 0xE for the data cache). Doing this resets the valid bit of the **addressed** Cache line (addressed by iu_dva[12:04] in the data cache).

6.10 Data Cache Protection checks

The Data Cache tags also incorporate 3 access protection bits (acc[2:0]), for checking access violations. These bits detect a protection or privilege exception in the W stage, so that protection traps can occur in W. This decouples the virtually addressed Data Cache from the TLB for a lot of cases. Ld/St instructions which hit in the Cache do not need the corresponding TLB entry to be present in the TLB (although St's do need a translated Physical address when they are ultimately drained from the WRB to main memory). If a ST instruction creates a Protection violation, the corresponding Data Cache line is invalidated. This is needed because the protection check signal is slower than the write to the Data Cache.

6.11 Cacheability of Memory Accesses

Pages that are declared as non-cacheable (C=0 in the PTE) are not cached in the Data Cache. For data consistency and implementation reasons, the following operations are also not cached.

Accesses when the MMU is disabled and alternate cacheability is disabled (EN, AC bits of the MMU PCR=0).

Accesses to any ASI except 0x8, 0x9, 0xA and 0xB.

Accesses to any non-memory physical address (as defined in the microSPARC-II MMU spec).

Accesses while the Data Cache is disabled (DE bit of the MMU PCR=0).

Accesses while alternate cacheability is disabled (AC bits of the MMU PCR=0).

Accesses while in Boot Mode.

Accesses to physical address spaces (PA[30:28]) 0x1, 0x3, 0x4, 0x5, 0x6, 0x7.

Note:Local Graphics space (PA[30:28]==0x2) may be cached.

Accesses by the MMU during tablewalks.

It should be noted, that a ST instruction to a non-cached address in ASI space 0x8, 0x9, 0xA and 0xB, invalidates the corresponding Data Cache line. This is because the ST has already updated the Data RAM by the time the cached/non-cached information is available. (This information is usually in the MMU. eg. TLB PTE.C bit etc.).

6.12 Data Cache Streaming

When the first half of the Data Cache line is brought back from main memory, the IU is released by the Data Cache controller for both Load and Store instruction misses. During the window (from the time the first half of the cache line is back until the second half of the cache line is filled), most instructions in the IU are allowed to proceed / stream, except for the following:

Ld / St instructions in any ASI space other than 0x8 / 0x9 / 0xA / 0xB.

Ld / St instructions which try to load or store from / to the second half of the **missed** cache line.

Any instruction one cycle after a parity error is detected on a fill.

A store instruction 1 cycle before the second line fill cycle, due to resource conflict, (both the IU and the Data Cache controller are trying to write the Data Cache RAM).

It is to be noted that stores which continue execution during the miss, only can do so due to the 4 deep write buffer. If the write buffer becomes full during this streaming, then the pipeline is held.

6.13 PTE Reference Bit Clearing

Many Paging based Operating Systems use the Referenced bit (R bit) in the PTE to approximate LRU behavior in accessing frequently used pages quickly. Clearing the R (Referenced bit) bit of a PTE could be costly in microSPARC-II because microSPARC-II has Virtual Caches (hence clearing the R bit of a PTE entails flushing that page from the I and D Caches) and no flash Clear instruction. The cost of flushing is two-fold:

1. The cycles spent in flushing each line of the cache. (Flash clear would have helped here).

2. The loss of cycles due to 'extra' cache misses due to cache line invalidations. (Flash clear does not help here).

To avoid both of these problems, **do not flush the I/D cache when the R bit is reset.**

As we shall see, this could mean that some recently referenced pages could get thrown out unnecessarily, but this should not happen very often in practice. This scheme nearly approximates the original LRU behavior, and it works on the following paradigm:

If a page is frequently accessed, then it will generate **at least 1** cache miss which is enough to make the MMU do a tablewalk (and hence set the R bit back to 1), before the Operating System daemon re-examines the R bit. This is based on the observed phenomenon that on an average, if we reset the R bit of a page, and this page is frequently accessed, then there will be at least 1 cache miss to that page which will force the MMU to set the R bit back to 1 quickly.

6.14 Powerdown & Parity Errors

The Data Cache RAM and TAG are both powered down during cycles when they are not used by the Data Cache controller in order to conserve energy dissipated by the microSPARC-II Chip. Powerdown is triggered either by a signal external to the Data Cache controller, or internally by the Data Cache controller state machine. Externally, the Data Cache controller follows a simple two way handshake protocol of request/grant to go into powerdown mode. The Data Cache controller also holds the IU pipeline during this period. For more on this please refer to the Powerdown chapter of this microSPARC-II document. Internally the Data Cache controller goes into powerdown mode during various State machine states, when the Data RAMS and TAG's are **both** not needed. This is because the RAM's and the TAG's both share the same powerdown control signal.

Parity errors on Data Cache fill, will invalidate only that particular cache line, during whose fill, the parity error took place. Parity errors during Non-cached misses do not cause any invalidations.

6.15 Diagnostic Strategy

Sublines and cache tags may be both read and written using ASI 0xF and 0xE respectively as previously discussed. The Data Cache will be structurally tested via the JTAG controller test ports. All register bits within the Data Cache and Data Cache tag are accessible via scan; on the chip level, all locations of these RAMs may be read or written by appropriate sequences of scan operations.

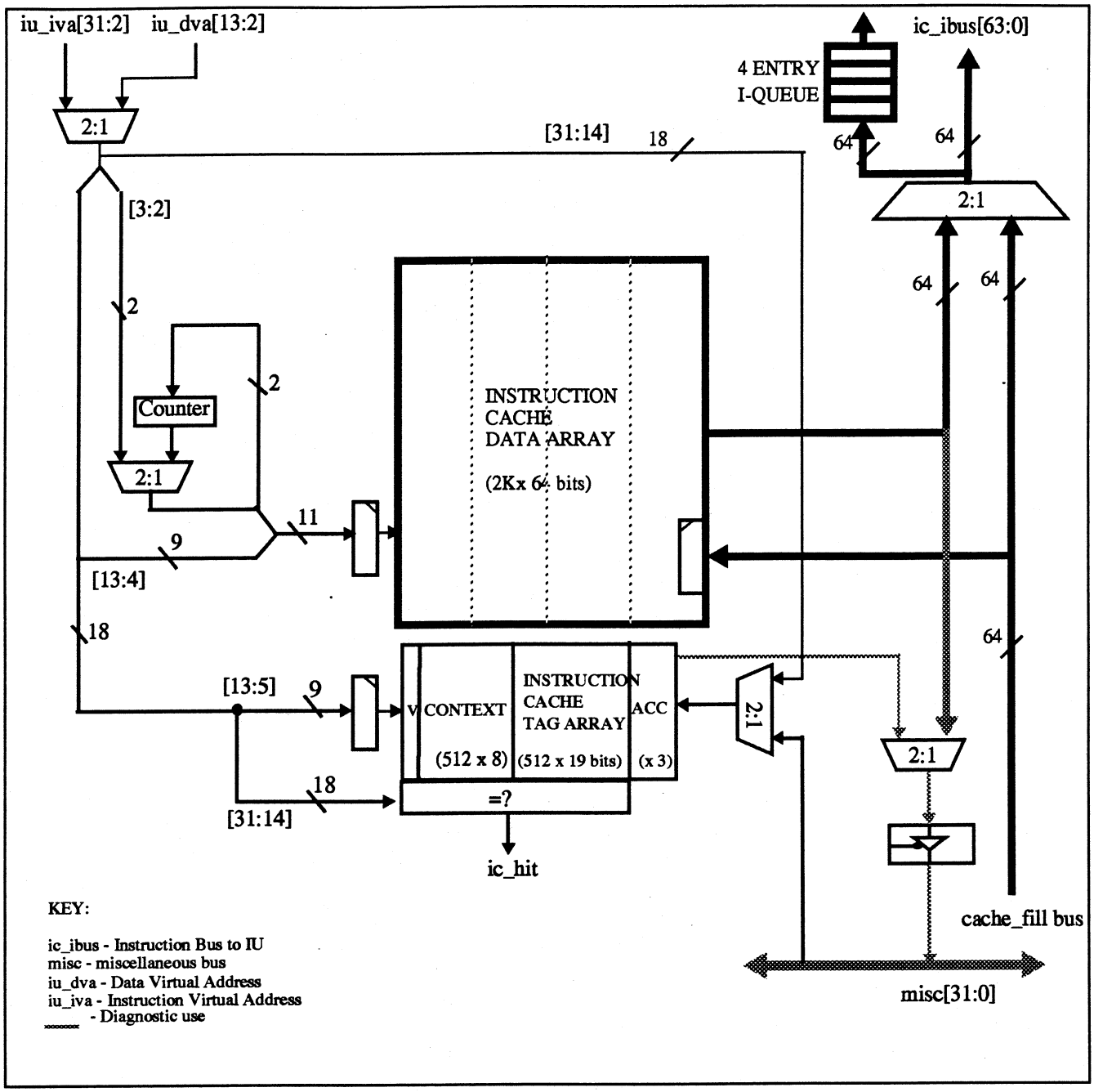
Chapter 7 I cache

7.1 Overview

The microSPARC-II Instruction Cache is a 16K-Byte, direct mapped cache, used on instruction fetch accesses from the CPU to cacheable pages of main memory. It is virtually indexed and virtually tagged. The instruction cache is normally addressed by iu_iva[13 :0]. The instruction cache is organized as 512 lines of 32 bytes of data. Each line has a cache tag store entry associated with it. There is no sub-blocking. On an instruction cache miss to a cacheable location, 32 bytes of data are written into the cache from main memory. The Instruction cache tags contain a 18 bit IVA[31:14] tag field, a 8 bit context field, 3 bit ACC field, and 1 valid bit.

Within the instruction cache block there are also cache bypass paths. These paths are used for noncached instruction fetches, and for streaming instructions into the IU on cache miss. A simple block diagram follows.

Figure 7.0 - Instruction Cache Block Diagram



7.2 Instruction Cache Data Array

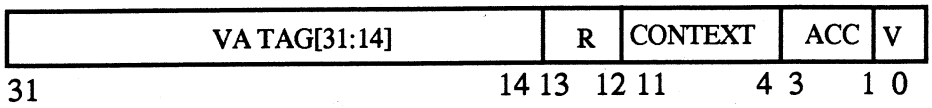
Diagnostic software may read and write the instruction cache directly by executing SingleWord load or store alternate space instructions in ASI space 0xD. Virtual address bits iu_dva[13:2] will be used to address the instruction cache in this mode; all other virtual address bits(addresses rollover), as well as the Context ID bits, ACC bits and the Valid bit are ignored during these operations.

The internal misc[31:0] data bus is used as input/output for ASI operations, and the cache_fill[63:0] bus is used to fill the Instruction cache on instruction cache misses.

7.3 Instruction Cache Tags

A instruction cache tag entry consists of several fields as follows.

Figure 7.1 - Instruction Cache Tag Entry



Field Definitions:

Virtual Address Tag - This field contains the virtual address of the data held in the cache line. The Instruction Cache Controller writes this field from bits [31:14] of the virtual address (iu_iva) of the line.

Acc bits - These 3-bit field indicates various levels of protection for that cache line. The field is copied from the TLB contents. (See Table 9, in Chapter 5: MMU).

Context bits- These indicate the context of the particular cache line. They are filled from the TLB.

Valid - This bit indicates that the line contains data. This bit is set when a cache line is filled due to a successful cache miss; a cache fill which results in a memory parity error will leave the Valid bit unset. A Flush instruction will clear the valid bit of the single line which is addressed by iu_dva[13:5] (only if the tag for the addressed line matches the flush address).

There are two input sources to the instruction cache tag array. The Virtual Address bits needed for the tag are used for cache updates due to instruction cache misses. The misc bus is used as input for alternate store operations.

Diagnostic software can read and write the instruction cache tags by executing word-length LDA and STA (Load and Store Alternate) instructions in ASI space 0xC.; dva bits [13:5] will select one of the 512 tags; all other address bits are ignored.

7.4 Instruction Hit/ Miss

The memory block size of data fetched from memory on instruction cache misses is 32 bytes. Memory will always return 32 bytes of data, starting with the requested doubleword first followed by the three remaining doublewords (even doubleword, then odd doubleword), which will wrap around a 32 byte boundary until the entire 32-byte

block has been returned. The transfer rate is one doubleword every 4/5 cycles from memory (one doubleword, then 3/4 dead cycles). The Cache array is written during the cycle that each word appears on the cache_fill bus[63:0] The following table illustrates the fill operation showing the order that words are written into the cache. Depending on the speed selection "sp_sel" input to the microSPARC-II chip, there will be a gap of some (usually 3 or 4) internal clocks in between every two words filled into the Cache.

Table 45 - Instruction Cache Fill Ordering

Requested Word	Order of fill
0	0, 1, 2, 3, 4, 5, 6, 7
1	1, 0, 2, 3, 4, 5, 6, 7
2	2, 3, 4, 5, 6, 7, 0, 1
3	3, 2, 4, 5, 6, 7, 0, 1
4	4, 5, 6, 7, 0, 1, 2, 3
5	5, 4, 6, 7, 0, 1, 2, 3
6	6, 7, 0, 1, 2, 3, 4, 5
7	7, 6, 0, 1, 2, 3, 4, 5

During an instruction cache fill, instructions from the missing line can be supplied to the IU or FPU by two separate mechanisms; these mechanisms are collectively called "streaming". In the first type of streaming ("bypass streaming"), instructions are bypassed around the cache data array to the IU / FPU in the same cycle that the array is being written - this can occur in all clock cycles of the fill sequence except the gap cycles. The second form of streaming ("gap streaming") occurs only during the gap cycles; any instruction word, **from any line in the cache** which has already been written into the RAM array can be accessed by reading the array. In a given cycle, the IU is able to accept the instruction word which it needs, immediately and instruction words which it **may** need in the future (prefetching). If, in a given cycle, the IU is requesting a word which is available via streaming, then that word is supplied to the IU and the pipeline is allowed to advance. The concept of streaming does not apply to non-cached instructions, as the IU does not have to be held for a cache fill.

7.5 IASI Bus Interface

The instruction cache block interfaces to the misc bus for ASI operations. Data from the misc bus to the Instruction cache comes from the writebuffer. There are control signals from the MMU to indicate when data on misc[31:0] is to be loaded into the Instruction Cache.

7.6 ICache fill Bus Interface

The instruction cache is filled by a separate 64bit bus (cache_fill[63:0] bus), from main memory. Keeping this bus separate from the other bus (misc[31:0]) on the chip, gives the flexibility to tune this bus for higher performance. Since this is a 64-bit bus (rather than 32 -bit), it gives the microSPARC-II caches, a shorter cache fill latency.

7.7 IU Instruction Bus Interface

The instruction cache block drives the IU instruction bus (ic_ibus). Instructions to the IU or FPU are sourced from either the latched value of the cache_fill bus (for bypass-streamed instructions on instruction cache misses, and for non-cached instruction fetches) or the instruction cache data array (for instruction cache hits, and for dead-cycle streamed instructions on instruction cache misses). The IU also fills the I-queue from this bus(ic_ibus).

7.8 Instruction Cache Flushing

The instruction cache tags are implemented with all the five flush mechanisms, (Page, Segment, Region, Context and User) as suggested in the SPARC reference MMU document. and it is activated by a word length alternate store instruction to ASI 0x10 - ASI 0x14. The IFLUSH instruction also can be used to flush the instruction cache. In both the cases the **addressed** (addressed by iu_iva[13:05]) instruction cache line's valid bit is reset by this operation, only if the corresponding tags match(The match criteria is determined by the **type** of flush instruction). The Instruction Queue is **not** flushed on an Instruction Cache flush because the maximum depth of the Instruction Queue is only 4 instructions and the IU disables any more Instruction fetches when it decodes an Instruction Cache flush opcode in the D-Stage.(The SPARC Architecture Manual allows 5 instructions after an Instruction Cache Flush instruction, for the IU to make the IU pipeline, the Instruction Queue and the Instruction Cache consistent). Again as in the data cache, the instruction tag diagnostic ASI (0xC) can also be used to reset the valid bit.

It is **recommended** that the Instruction Cache be flushed whenever the referenced bit(R bit) of any cacheable line is reset in the corresponding entry in the PageTables.

To maintain consistency it is **required** that the software flush the Instruction Cache whenever the ACC bits or the C bit of a cacheable location is changed in the corresponding entry in the PageTables.

7.9 Cacheability of Memory Accesses

Pages that are declared as non-cacheable (C=0 in the PTE) are not cached in the instruction cache. For data consistency and implementation reasons, the following instruction fetch operations are not cached(regardless of the PTE.C bit).

Accesses when the MMU is disabled and alternate cacheability is disabled (EN, AC bits of the MMU PCR=0).

Accesses while the instruction cache is disabled (IE bit of the MMU PCR=0).

Accesses while in Boot Mode.

Accesses to sources in physical address spaces 1-7.

7.10 Diagnostic Strategy

Sublines and cache tags may be both read and written using ASI 0xD and 0xC respectively as previously discussed. The instruction cache will be structurally tested via the JTAG controller test ports. All register bits within the instruction cache and instruction cache tag are accessible via scan; on the chip level, all locations of these RAMs may be read or written by appropriate sequences of scan operations.

Chapter 8 Memory Interface

8.1 Overview

MicroSPARC-II architecture allocates 256MB of space for the system memory (Physical address space'0', defined by mm_pa[30:28]), and the actual memory interface and the memory management unit can also support up to 256MB.

The following sections describe the general memory layout for the microSPARC-II based system and then continue to explain each of the logical blocks within the Memory Interface block.

The microSPARC-II Memory Interface block is logically divided into three subsections, the Memory Control Block (MCB), The Data aligner and Parity check/generation logic (DPC) and the Ram Refresh control (RFR).

8.2 Memory Subsystem

The interface is designed with the following criteria in mind:

- 64 bit Data bus to increase memory bandwidth.
- 1 bit parity per word (32 bits) for reduced cost.
- Memory divided into blocks which can support different density devices. This will allow relatively small memory increments with a small number of blocks.
- Allow for future higher memory requirements by supporting next generation of DRAM devices.

Typically a carefully laid out system board using the microSPARC-II chip would require 60ns, 3.3V/5V DRAMs at 70MHz clock speed. The designer however, should use the memory interface AC specifications in the microSPARC-II datasheet, to select the appropriate DRAM speed for a specific system and clock speed.

8.2.1 Memory Organization

MicroSPARC-II architecture defines a 28-bit physical address space for memory (PAS 0). This means a 256MB block for system DRAM.

This 256MB is divided into 8 banks, each capable of addressing up to 32MB. The banks are defined as follows:

- Each bank is selected by a separate RAS line. There are a total of 8 RASes for DRAM banks (ras_l[7:0]).
- The banks have a 64bit data path to microSPARC-II.
- Banks 0, 2, 4, 6 use the same 2-bit CAS lines (cas_l[1:0]), to select the upper or lower 32 bits (high or low word).
- Banks 1, 3, 5, 7 use the other 2-bit CAS lines(cas_l[3:2]) to select the upper or lower 32 bits.
- All the banks use the same write signal (mwe_l).
- All the banks use the same 22-bit multiplexed Row/Column address bus. At the time of finalizing the microSPARC-II memory interface, DRAM manufacturers were proposing 2 addressing schemes for 4Mx4 devices, an 11-row/11-column and a 12-row/10-column. Although the 4Mx33 Simms will use the DRAMs based on 11x11 matrix (to allow the use of a 4Mx1 DRAM for parity). microSPARC-II also provides a 12th DRAM address bit, which would allow the 12x10 matrix DRAMs to be used with microSPARC-II chip in other applications.

The memory interface is designed with the 4bit wide DRAM devices in mind. Using 16 such devices (or 2 SIMMs with eight devices on each) will provide the required 64bit wide data bus. In addition, each bank will require two 1bit wide devices of the same depth (If using SIMMs, one on each SIMM) to store the 2 parity bits.

Hence, each bank can be populated using one of the following configurations:

- 8MB (1Mx64) of data, using 16 of 1Mx4 devices for data and 2 of 1Mx1 for parity, or using 2 of 1Mx33 SIMMs.
- 32MB (4Mx64) of data, using 16 of 4Mx4 devices for data and 2 of 4Mx1 for parity, or using 2 of 4Mx33 SIMMs.
- 16MB (2Mx64) of data, using 8 of 2Mx8 devices for data and 2 of 2Mx1 for parity, or using 2 of 2Mx33 SIMMs.

Note that a pair of double-density (e.g. 16Mx33) SIMMs will occupy 2 banks (Need 2 RASes).

8.2.2 Access to Unused or Unpopulated Memory regions

Similarly, if a bank contains less than the defined maximum of 32MB, the real memory will be mirrored on the higher unused portions and an access from any of the unused sections will be mirrored to the corresponding location in the lowest block and no errors will be

generated. For example, if a bank contains 8MB of real memory, this will be mirrored on the remaining 3 empty portions.

However, an access from a fully unused (empty) bank will complete, but its result will be unknown and may cause a parity error.

8.3 Memory Control Block (MCB)

The operations that occur on the memory bus are data reads, writes, and read-modified-writes required for cpu execution, instruction fetches and prefetches, translation buffer accesses during table walks, reads and writes by IO devices, and all RAM refresh. The Memory Control Block (MCB) keeps track of the priorities of memory operations and completely controls the DRAM based main memory.

As shown in the following diagram, MCB contains 2 major logic blocks, labeled "ASM" and "ADEL" which perform the memory arbitration and address mapping functions respectively. This blocks will be described in the following subsections. MCB also includes some input and output register blocks, which provide the synchronization among input and output signals.

8.3.1 Arbitration State Machine (ASM)

ASM is responsible for detecting the requests from MMU and Refresh blocks, arbitrate between them if necessary and grant the appropriate request. Once a request is granted, the MCB will carry out the requested memory operation which will consist of one or more memory cycles. The following table lists all the types of memory operations performed by the MCB, the possible request sources and the type and number of cycles involved.

Figure 8.0 - MCB block diagram

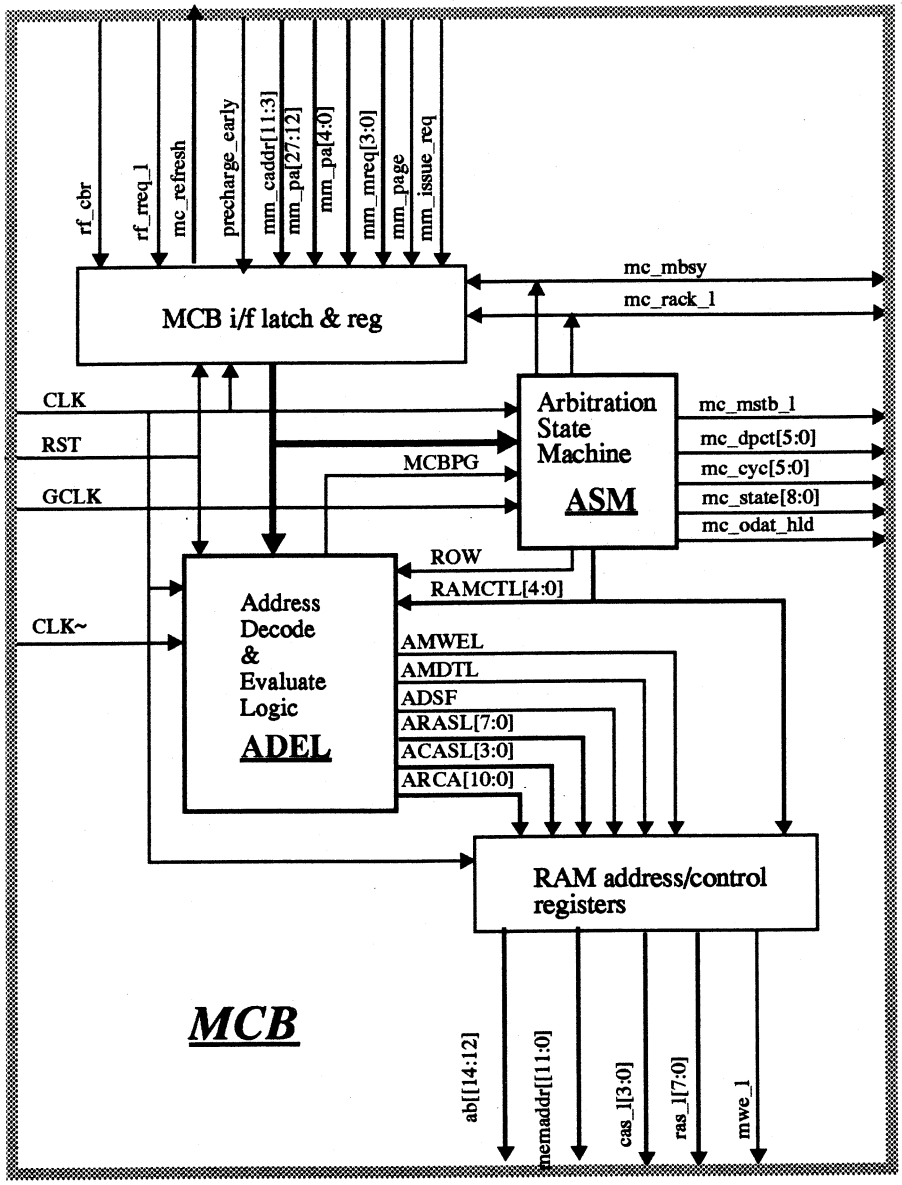


Table 46 - Memory operations performed by MCB

Operation	Source	Memory Cycles produced
d.rd.32b	MMU. Used to fill one line of I-cache (Inst-Fetch) or do an SBus burst read of 32 bytes.	32 bytes are read from DRAM in a single operation, using 4 longword (64bit) read cycles. The first read is paged or non-paged, from the address given on PA. The following 3 reads are paged. ADEL will supply the address for the next 3 reads, incrementing or wrapping it as necessary, in order to read a 32 byte aligned block and fill a whole I-cache line.
d.rd.16b	MMU. Used to fill one line of D-cache or do an SBus burst read of 16bytes.	16 bytes are read from DRAM in a single operation, using 2 longword (64bit) read cycles. First read is a paged or non-paged cycle, using the address supplied on PA. The next cycle is a paged read, where ADEL will increment or wrap the address in order to read a 16byte aligned block from memory.
d.rd.8b	MMU. Used for IU and SBus longword reads.	8 bytes are read from DRAM, using a paged or non-paged longword read from the address supplied by PA.
d.wr.8b	MMU. Used for IU and SBus longword writes.	8 bytes are written to DRAM, using a paged or non-paged longword write to the address supplied by PA.
d.wr.4b	MMU. Used for IU and SBus word writes.	4 bytes are written to DRAM, using a paged or non-paged word write to the address supplied by PA.
d.rmw.2b	MMU. Used for IU and SBus byte writes.	a halfword (16bit) write to DRAM in a single operation, using a paged or non-paged word read followed by a paged word write, using the same address supplied by PA. MCB will perform the read and write cycles and will instruct DPC to latch the 16bit write-data from the source, insert it in the appropriate halfword of the word read from memory and then gate it back on memory data-bus as the write data.
d.rmw.b	MMU. Used for IU and SBus byte writes.	a byte (8bit) write to DRAM in a single operation, using a paged or non-paged word read followed by a paged word write, using the same address supplied by PA. MCB will perform the read and write cycles and will instruct DPC to latch the 8bit write-data from the source, insert it in the appropriate byte of the word read from memory and then gate it back on memory data-bus as the write data.
cbr.ref	RFR. Used to do a refresh cycle on all DRAM	Will force a Cas-before-Ras refresh cycle to be performed on all DRAM banks with four banks refreshing at the same time.
d.wr.16b	MMU. Used for SBus 16 bytes burst write	16 bytes are written to DRAM in a single operation, using 2 longword (64bit) write cycles. First write is a paged or non-paged cycle, using the address supplied by PA. The next cycle is a paged write, where ADEL will increment..

8.3.2 Arbitration for Memory Access and ASM Priority Scheme

ASM arbitration scheme is based on the following rules:

- All requests are checked at the end of each operation (for multi cycle operations, this means the end of last memory cycle) and:

  If: no requests are pending, ASM will enter the idle state and will remain there until a request is detected.

  If: only one request is pending, it will be granted and the operation will begin.

  If: More than one request is pending, the one with the highest priority will be granted and the operation will begin. The priorities are as follows:

  h) MMU is the highest priority, except when the current cycle is also an MMU request, in which case it will be considered the lowest priority. This is to prevent bus locking as a result of back to back MMU requests.

  i) RFR has the lowest priority, except when the current cycle is an MMU request, in which case it will have higher priority.

  If: While in idle, an RFR request is detected, the state machine will advance to a "Check" state, where it'll look to see if an MMU request occurred just as RFR request was accepted. If there are no MMU requests, ASM will continue to acknowledge the RFR request and do the cycle, else, it will do the MMU cycle.

8.3.3 Address Decode & Evaluate Logic (ADEL)

This block primarily monitors the address and function-select signals coming from MMU and RFR and performs the necessary decode and re-mapping of the memory address and control lines. Based on commands received from ASM, ADEL gates the low address (bits [4:3] which is combined with the rest of the address bits driven from MMU) and memory control signals required for the current operation out to memory.

The mapping of system memory is discussed in the following section.

8.3.4 Address Mapping For System DRAM

From the 31 bits of the physical address bus driven by MMU block (mm_pa[30:0]), the three MSBs (mm_pa[30:28]) represent 1 of the 8 physical address spaces (PAS) as defined in microSPARC-II architecture. From these, only PAS0 is of concern to MCB, since an MMU request from MCB will only be made if an access to system memory is required. Hence ADEL ignores the mm_pa[30:28] bits.

When a memory cycle request is detected, ADEL uses the mm_pa[27:02] address bits to complete its decode. The following table describes the decode scheme used for system memory.

A maximum of 1024 memory cycles can be made from a contiguous block, while remaining within a DRAM page. This gives a maximum of 8K (1024x64) block size which can theoretically be accessed using page mode cycles only.

A point to note from the table below, are the staggered decoding of mm_pa[24:21] for memaddr[10:9]. This was necessary in order to allow different size devices (1Mx4 and 4Mx4) to be used while maintaining the largest common contiguous block, which is dictated by the least dense device.

Also, as shown in the table, mm_pa[23] is used as both memaddr[10] for column address and memaddr[11] for row address. This is to cater for the 2 different 4Mx4 DRAM architectures, 11x11 matrix and 12x10 matrix.

Table 47 - Physical Address decode for System Memory

PA	Decode
30-28	Not used. System memory limit is 256 MB.
27-25	Decode to select 1 of 8 RASes: 000 RASL0 1st 32MB bank. 100 RASL4 5th 32MB bank. 001 RASL1 2nd 32MB bank. 101 RASL5 6th 32MB bank 010 RASL2 3rd 32MB bank. 110 RASL6 7th 32MB bank 011 RASL3 4th 32MB bank. 111 RASL7 8th 32MB bank
24	Decoded as row address bit 10 (memaddr10). Required for 16MBit DRAMs.
23	Decoded as column address bit 10 (memaddr10) and row address bit 11 (memaddr11). Required for 16MBit DRAMs. See text for more information.
22	Decoded as row address bit 9 (memaddr9). Required for 4MBit DRAMs.
21	Decoded as column address bit 9 (memaddr9). Required for 4MBit DRAMs and up.
20-12	Decoded as row address bits 8 to 0 (memaddr[8:0]). Required for 1MBit DRAMs and up.
11-3	Decoded as column address bits 8 to 0 (memaddr[8:0]). Required for 1MBit DRAMs and up.
2	Decoded to select one of 4 CASes: 0 CASL0 Lower address word (memdata[63:32]) for banks 0,2,4,6. (bit 25 = 0 1 CASL1 Higher address word (memdata[31:0]) for banks 0,2,4,6. (bit 25 = 0) 0 CASL2 Lower address word (memdata[63:32]) for banks 1,3,5,7. (bit 25 = 1) 1 CASL3 Higher address word (memdata[31:0]) for banks 1,3,5,7. (bit 25 = 1)

PA	Decode
1-0	Not used for external decode. Byte and halfword writes are achieved by MCB and DPC doing a read, update, write sequence. This bits are used then, to select the appropriate data fields.

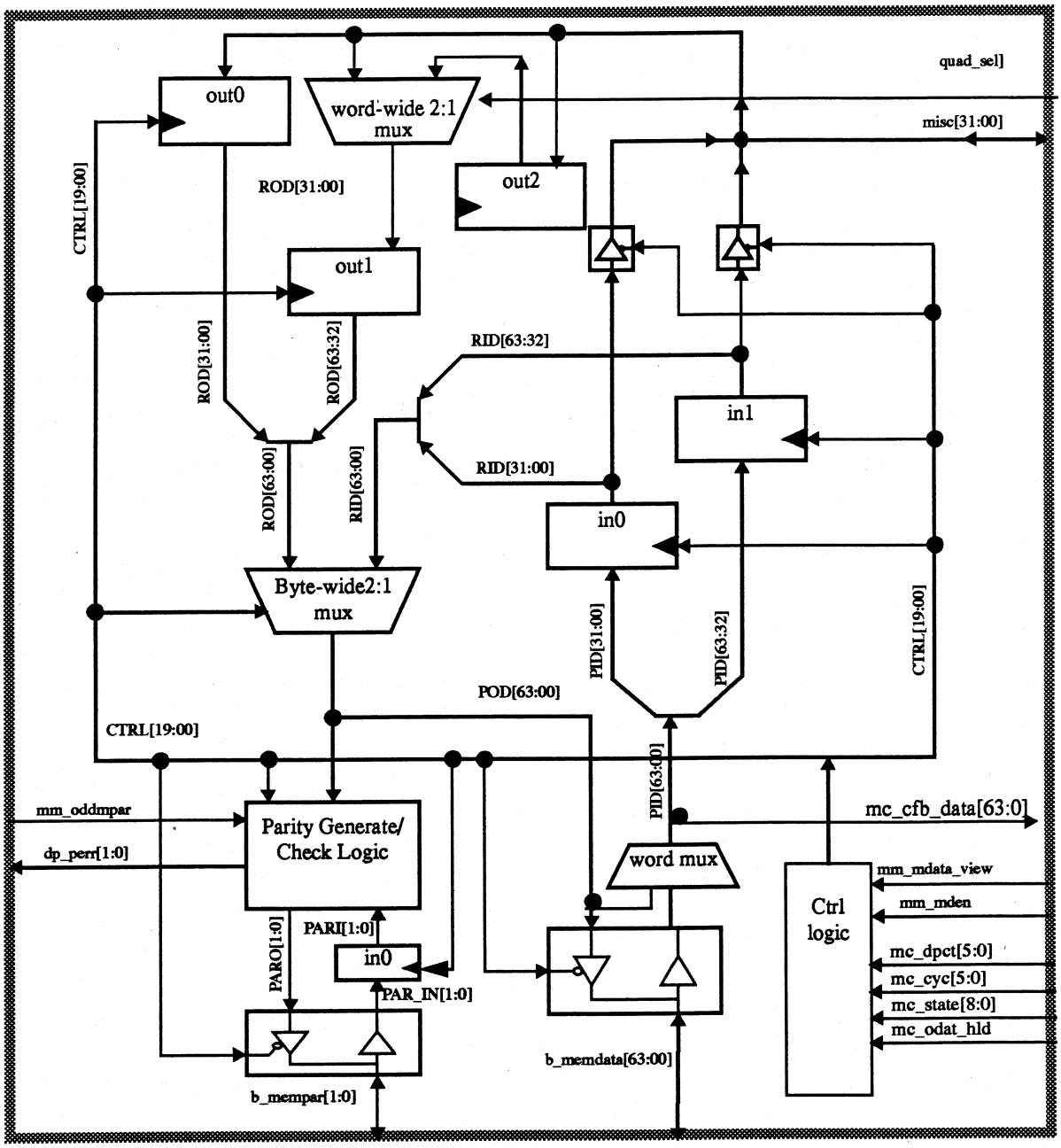
8.4 Data aligner and Parity Check/ generate logic (DPC)

DPC is responsible for transferring data between external memory data bus and the internal data path as well as generating and checking of parity for system main memory (DRAM).

During any read, write or hardware controlled read-modify-write cycle, DPC will perform the necessary data alignment and byte/halfword placement. It will also provide temporary storage for hardware controlled read-modify-write cycles, resulting from byte/halfword write cycles to memory.

DPC also contains the parity generation and checking logic. The parity is composed of 1 bit per word (32 bits) and is used for system DRAM only.

Figure 8.1 - DPC Datapath and Parity Control Block Diagram



The type of parity operation for the system DRAM is determined by the state of the Parity Control bit (PC) in the Processor Control Register as described in the following table:

Table 48 - Parity Control Definition

PC	Description
0	Check/Generate even Parity.
1	Check/Generate odd Parity.

Since system parity is 1-bit per word, any byte or halfword store operation, will result in a hardware controlled read-modify-write cycle. During the read part of such operation, the word parity will be checked and if an error is detected, a parity error will be generated. After the word has been updated to contain the new byte/halfword, a write operation will be performed, which will also update the parity. Mempar[0] is associated with memdata [31:0] while mempar[1] is assocaited with memdata [63:32].

The flow of data and type of operations performed by DPC are governed by the Memory Control Block and the commands it receives from MCB.

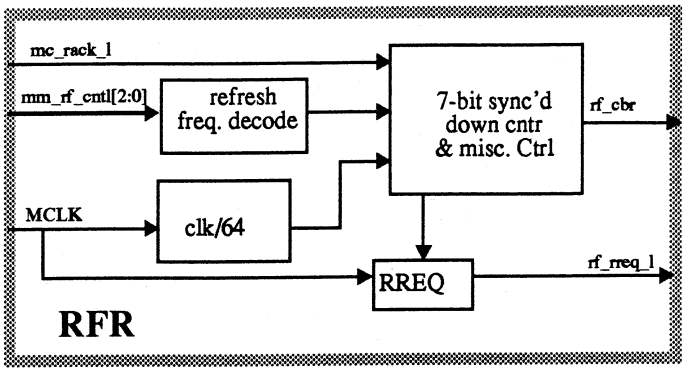
DPC block diagram, given here, shows the basic data paths connecting the 64bit external memory bus (b_memdata[63:00]) to the 32bit internal one (misc[31:00]). The parity check/generation logic is shown to be on the output path, but for input data, parity is checked after it is clocked into the registers and gated through the alignment mux.

The alignment mux is also used to combine and produce the output data during a read modify write sequence. The complexity of this mux is reduced by having the byte or forward data which is to be written to memory, already in the correct position. This is done by the block sourcing the data on misc (D or I cache, IU, SBus controller).

8.5 RAM Refresh Control (RFR)

The refresh control logic (RFR) is a simple request generator, asserting a request to MCB at fixed intervals. MCB will service this low priority request by performing a Cas-before-Ras type refresh cycle on all system RAM. Banks 0, 2, 4, 6 and banks 1, 3, 5, 7 will have RAS's asserted at different cycle to reduce current spike.

Figure 8.2 - RAM Refresh Control block diagram.



RFR refresh rate can be selected by programming 3 bits of the Processor Control Register according to the following table. These bits are then passed to RFR as mm_rf_cntl[2:0] input bits, which controls the rf_rreq_l rate.

microSPARC-II supports TOSHIBA TC514900AJLL/AZLL DRAM for self-refresh. After seeing assertion of mm_rf_cntl[3] to 1 , memory controller will set states for self-refresh and enters the power-off mode in 2 us provided that there is no outstanding Local Graphics operation. The user needs to make sure any Local Graphics operation has completed before the assertion of mm_rf_cntl[3]. After the 2 us waiting period, the user can either turn off power to microSPARC-II or write mm_rf_cntl[3] back to 0 to continue CPU's operation. Note that during the 2 us waiting period, the user cannot change the mm_rf_cntl[3] bit nor turn off power to microSPARC-II. Once microSPARC-II enters the power-off mode, it will stay in that mode until the chip is reset. The system should keep the signal levels of CAS and RAS, (preferably address also) on the board before and after power to microSPARC-II is turned off. The system is prohibited to issue any more memory request once mm_rf_cntl[3] is asserted.

Table 1: Refresh Rate Control bits.

mm_rf_cntl [2:0]	Refresh interval
0 0 0	Assert a refresh request once every 128 CLK periods. With this setting, adequate refresh is guaranteed for CLK values of down to 8.6MHz. This is the default after power up.
0 0 1	No Refresh!
0 1 0	Assert a refresh request once every 704 CLK periods. With this setting, adequate refresh is guaranteed for CLK values of down to 48MHz.
0 1 1	Assert a refresh request once every 896 CLK periods. With this setting, adequate refresh is guaranteed for CLK values of down to 60MHz.
1 0 0	Refresh every 1216 CLK periods to run above 83 MHz.
1 0 1	Refresh every 5120 clocks for low refresh DRAMs.
1 1 0	Refresh every 1408 CLK periods to run above 100 MHz.
1 1 1	Refresh every 1792 CLK periods to run above 125 MHz.

The RFR is also responsible for initializing the DRAMs on power-up.

After power-up and before they can be reliably used, DRAMs require a 200us "Wait" period followed by 8 Cas-before-Ras refresh cycles.

For systems built around microSPARC-II, the reset must remain active for at least 200us after power-up, to satisfy the "Wait" period. Systems built around microSPARC-II should guarantee an active reset duration of 200 us or more.

After an active reset, the "mm_rf_cntl[2:0]" bits which reside in the MMU's PCR register are set to "000" (See table above), setting RFR to generate a refresh request every 128 clocks. In addition, RFR itself, asserts its "rf_cbr" and "rf_rreq_l" signals, forcing MCB to enter a "cbr" state, where it will perform 8 Cas-before-Ras refresh cycles, completing the DRAM initialization cycle. After that, RFR will negate both "rf_cbr" and "rf_rreq_l" signals, allowing MCB to proceed to its normal operation state.

8.6 clock speeds

microSPARC-II memory controller is designed to operate over a variety of clock frequencies. There are four speeds available and controlled by pin sp_sel[1:0]. Namely, sp_sel = 00 for low speed; 01 for medium speed; 10 for high speed and 11 for ultra high speed. Wait states are inserted in medium speed compared to low speed; and high speed has even more wait states. For example, low speed has a read bandwidth of 4 cycle; medium speed, high speed and ultra high speed have 5, 6 and 7cycle read bandwidths, respectively. Timing around microSPARC-II is designed towards systems that use 60 ns DRAM for low speed to achieve 70 MHz; medium speed for 85 MHz; high speed for 100 MHz; and ultra high speed for 125 MHz. For systems prepared to use slower DRAM at 70 MHz. Selecting higher speeds can provide extra time to compensate for slower DRAM.

8.7 Summary of cycles

Here is a summary of number of cycles designed for different interface signals to the DRAM at various speed selects. Only cycles that are important to system usage are given here. The purpose is to provide the system designer with a quick reference to evaluate which kind of DRAMs may be suitable for the desired speed select choice. Please note that cycle numbers are given in terms of clock. Actual delays from clock to output of each pin may differ.

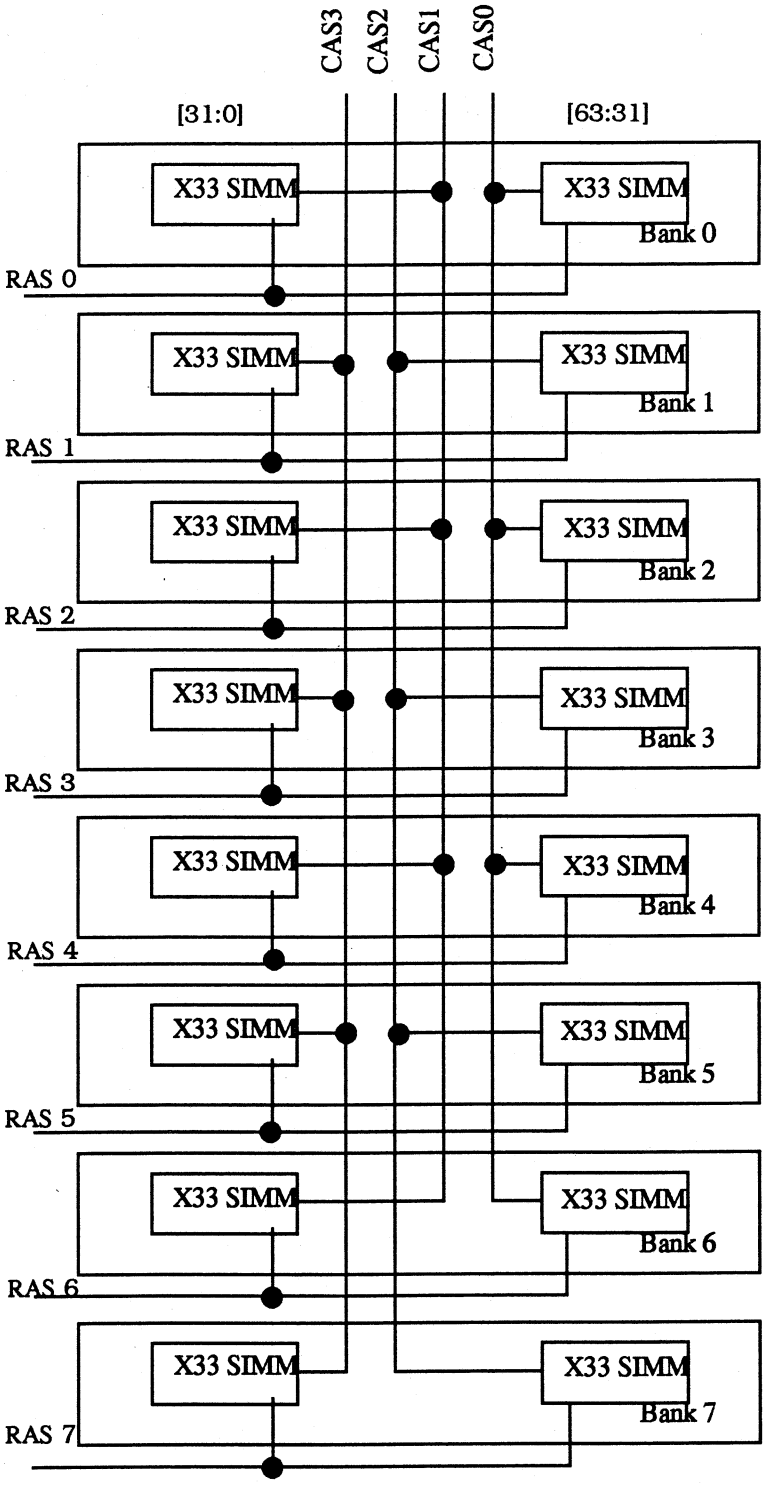
Table 2: Summary of cycles of selected parameters

	Number of cycles at sp_sel = 00	Nunmber of cycles at sp_sel = 01	Number of cycles at sp_sel = 10	Number of cycles at sp_sel = 11
RAS precharge	3.5	3.5	4.5	5.5
CAS precharge	1	1	2	2
RAS active (read)	6.5	7.5	8.5	10.5
RAS active (write)	5.5	5.5	6.5	8.5
CAS active (read)	3	4	4	5
CAS active (write)	2	2	2	3

Table 2: Summary of cycles of selected parameters

	Number of cycles at sp_sel = 00	Nunmber of cycles at sp_sel = 01	Number of cycles at sp_sel = 10	Number of cycles at sp_sel = 11
Data, WE, parity sent before CAS	1	2	2	2
Data, WE, parity hold after CAS active	2	2	2	3
Column address sent before CAS	1	1	2	2
CAS before RAS (refresh)	1.5	1.5	2.5	3.5
RAS active (refresh)	6.5	6.5	6.5	8.5

8.8 Memory configuration of RAS and CAS



8.9 Local Graphics Bus interface

MicroSPARC-II supports the Local Graphics interface as described in The Local Graphics Bus Description Rev 1.0 (Appendix A). For additional information, please refer to The AFX Bus Specification 2.1 delivered under Non Disclosure Agreement.

It uses the same data and address bus that connects to DRAMs. MicroSPARC-II allocates 256Mbyte address space for graphics access. This space is addressed by PA[30:28] being three bit of 010. GCLK is the graphics clock that is divided by 3 of internal system clock. The frequency range of GCLK in microSPARC-II is 23-42 MHz. All graphics pins have timing requirements with respect to GCLK. Various dead cycles must be inserted between different types of operation both within the graphic space and between graphics and memory space as they share the same data and address bus. There is a page-mode, and a non-page mode operation types depending on which address lines are changed from the previous access to the graphic space. Local Graphics is a slave-only device that accepts read and write instructions of sizes of byte, halfword, word or doubleword. The memory controller receives instructions from the processor or DMA. The processor issues read and write instrutions of sizes byte, halfword, word or doubleword. The memory controller also supports DMA issues of doubleword, 16 bytes or 32 bytes read; and all sizes of writes up to 16 bytes. DMA burst mode accesses are broken down into multiple doubleword Local Graphics accesses with no interruption in between. There will not be parity checking or generation associated with Local Graphics access.

The interface that connects to the data bus should have at least a 4 double-word deep FIFO for store buffering. To avoid potential ethernet time-out problem, microSPARC-II allows 2.0 us maximum operational latency for any Local Graphics instructions. Systems using Local Graphics interface should design for a worst case latency of 2.0 us, and an average latency of no more than 1.0 us. If the 2.0 us worst case reply latency is violated, the content of the DRAM cannot be guaranteed.

Here is a summary of what features the Local Graphics Bus interface supports:

- Non-cached direct processor access - including bytes, halfword, word and double word access.
- Cached direct processor access.

- Full DMA access - supports sizes up to 32 bytes that microSPARC-II supports. All non-burst mode DMA operations are required functionality. Burst mode operations are consistent with existing design.
- Page mode detection to Local Graphics access.
- Suppress parity checking.
- Hold off processor read access until slave's write FIFO has zero or one writes pending. Also hold off processor write access until a slot is available in the slave's FIFO.
- Timeout declared when no response from Local Graphics slave for 2047 GCLKs after the last request to Local Graphics was issued. Level 15 interrupt will be generated.
- Support bandwidth of one GCLK per write access; two GCLKs per read access.

Chapter 9 SBus Controller

9.1 Overview

The SBus Controller (SBC) is the I/O subsystem that handles input and output between local resources, including the CPU, system memory, and control space, and all external system resources. The SBC implements SBus in accordance with the IEEE P1496 SBus Specification.

The SBC supports:

- Programmed Input/Output (PIO) transactions between the CPU and SBus slave devices.
- Direct Virtual Memory Access (DVMA) transactions between SBus masters and System Memory or System control space or Local Graphics space (referred to as local DVMA).
- Direct Virtual Memory Access (DVMA) transactions between SBus masters and other SBus slave devices. (referred to as SBus -SBus DVMA)
- Variable speed: SBus clock is divided either by 3, 4 or 5 from the CPU clock. This allows the CPU frequency range from 50 - 125 MHz. Special logic is added to make the SBus interface transparent to this varying frequency, thus avoid the need for synchronizer which can hurt the performance significantly.
- 5 slave select lines. Address space 0x20000000 reserved for Local Graphics.
- Up to 32-byte burst transfer for local DVMA, 64-byte burst for SBus-SBus DVMA .
- CPU atomic cycles, but not DVMA atomic cycles
- Power management by indicating to the clock controller when it is safe to stop the processor clock.
- All standard SBus features: dynamic bus sizing, retries, bus arbitration, burst transfers, watchdog timer, and error reporting.

Interrupts and SBus Reset are not implemented in the SBC; these functions are handled elsewhere.

The SBC plays many SBus roles. It serves as an SBus controller by arbitrating bus requests, managing the translation of virtual addresses,

enabling slave cycles, etc. In addition, the SBC may act as either an SBus master or an SBus slave. For PIO transactions, the SBC acts as an SBus master. For DVMA transactions, the SBC can act as either a slave and controller or only as a controller, depending on the target of the DVMA transaction as indicated by the physical address. For local DVMA, the SBC has a role as both a bus controller and a slave device. For SBus-SBus DVMA, the SBC has a role as a bus controller only, not as a slave.

The SBus visible part of PIO transactions consists of an SBus slave cycle only; the address translation is done in advance of the bus acquisition.

PIO transactions occur when the CPU executes loads or stores to I/O (SBus) space. In the case of a PIO write transaction, the write is posted. Processing in the CPU continues while the SBus transaction completes in the SBC. A CPU stall will occur only if another PIO transaction is attempted before the previous PIO write transaction completes. In the case of a PIO read transaction, processing is always stalled until the data becomes valid at the end of the SBus transaction.

DVMA transactions occur when an SBus master has acquired the bus in order to execute a transaction to a slave. A DVMA transaction consists of an address translation cycle and a slave cycle. The target of the slave cycle is determined once the translation cycle is completed. The slave target can be either a local resource, defined as locations in either system memory, system control space or Local Graphics space, or another SBus device.

During the address translation cycle, the SBC obtains a virtual address from the DVMA master and submits it to the MMU for translation. The MMU returns a physical address. The type of DVMA slave cycle, either local or SBus-SBus, is determined from the physical address.

To improve I/O bandwidth, microSPARC-II pipelines the bus arbitration and bus grant with the previous DVMA write cycle . In that case, the SBC grants the bus to the next Master and receives the virtual address (if there is any bus request) at the time data from the previous DVMA write is being transferred to memory. This eliminates the 2 additional SBus clocks that would be required should the controller wait until the end of the data transfer to memory from the previous DVMA write. However, for an SBus benchmark using bus grant to calculate the SBus latency, it could look like the operation is slower.

A significant point concerning memory data transfers is that since system memory is a local resource, it is necessary for memory data to

pass through the SBC. Local DVMA slave cycles have two distinct, sequential (or pipelined) operations in the SBC: a data get followed by a data put operation. A data get operation loads up to 32 bytes of data into an internal data store. A data put operation transfers the data from the internal data store to a destination. The data get operation can either be an internal data transfer or an SBus cycle, depending on the read/ write direction. The data put operation will respectively be an SBus cycle or an internal data transfer. To improve I/O bandwidth, microSPARC-II implements special cases in which the data get and the data put operations are pipelined. This applies to DVMA write 32 bytes, in which the data is 16 byte aligned, i.e. pa[3:0] = 0 Hex, or DVMA read 16 / 32 bytes in which the data is 8-byte aligned i.e. pa[2:0] = 0.

Data Get and Data Put

	Data Get	Data Put
Read	Internal Data Transfer	SBus Cycle
Write	SBus Cycle	Internal Data Transfer

For local DVMA read slave cycles, an internal data transfer occurs during the data get stage, and an SBus slave cycle occurs during the data put stage. In this case, the data get operation shows up as a pause between the SBus translation cycle and the SBus slave cycle. If the condition for pipelining is met, the data put operation will start as soon as the first two words of the data-get are received by the SBC.

For local DVMA write slave cycles, an SBus slave cycle occurs during the data get stage, and an internal data transfer occurs during the data put stage. In this case, the DVMA transaction is finished on the SBus after the slave cycle completes the data get stage. The current cycle is not held up during the internal data transfer, but the data put stage will result in bus latency before the next translation cycle can occur. If the pipelining condition is met, the data put stage can start as early as the first four words in the data get stage are received by the SBC.

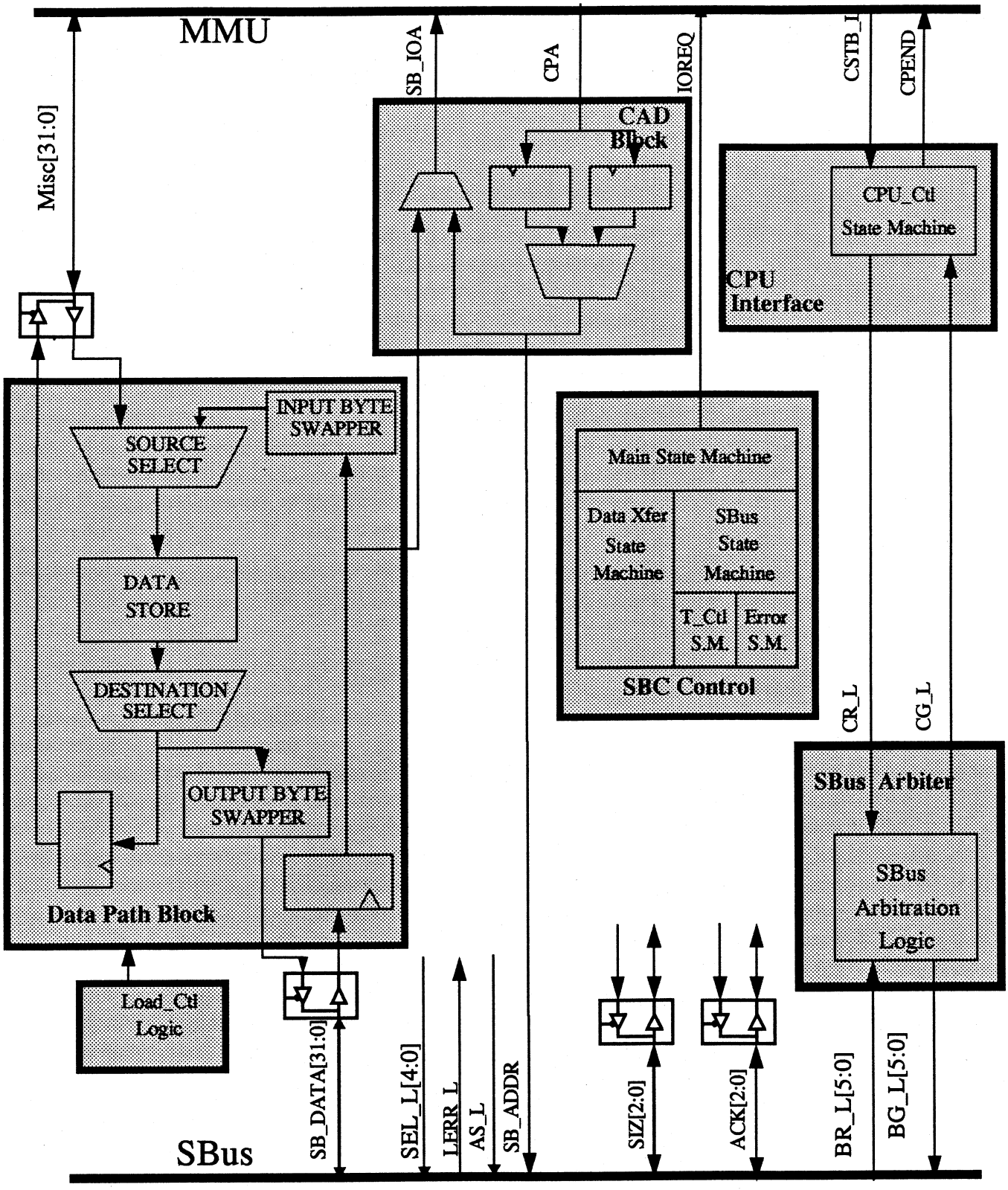
SBus-SBus DVMA slave cycles do not involve the SBC as a slave target. The data transfer is between an SBus master and another SBus slave. There is no data get and data put operations in this case.

As a bus controller, the SBC has to handle bus errors and watchdog timeouts. Bus errors that occur during PIO cycles are handled by making the current state of the bus cycle available to the MMU. Bus errors that

occur during DVMA cause the SBC to intercept the slave cycle from the intended slave target, and itself become the slave target in order to terminate the cycle with an error. Note that in case an error occurs whether due to a translation error , size not supported error, or parity error during data get of DVMA read, the SBC terminates the cycle with ERR ACK without asserting AS_L. Watchdog timeouts occur when an internal timer expires and the SBC terminates the slave cycle with an error. The timeout counter is incremented by the SBus clock and counts when AS_L is active. ERR ACK is asserted 256 SBus clocks later.

The sub-blocks of the SBC are: the CPU Interface, Address Steering, SBus Arbiter, Main Control, Data Transfer Control, SBus Slave and Target Control, Data Path and Control, and Error Control blocks. These sub-blocks are schematically shown in the following block diagram.

Figure 9.0 - SBus Controller Block Diagram



9.2 CPU Interface:

The CPU interface block handles the MMU / SBC handshake protocol, arbitrates for the SBus, catabolizes double word PIO into single word PIO, if appropriate, and supports dynamic bus sizing and bus cycle retries. The data sizes supported for PIO cycles are: byte, half byte, word and double word. There is also a high-performance feature that allows for very fast PIO writes to occur, which is especially important for certain operations that require fast output, such as graphics.

The CPU interface is double buffered, meaning that a copy of the entire state of the current cycle is retained for both PIO reads and PIO writes. The double buffering is necessary in the event of dynamic bus sizing, catabolic double word transactions or bus cycle retries. The buffering also permits a DMA address translation to occur concurrent with a PIO transaction. This is important in deadlock avoidance.

The deadlock could occur when simultaneous PIO and DVMA transactions occur. The deadlock is avoided by buffering the entire state of the PIO transaction, and allowing the DVMA transaction to proceed. Upon completion of the DVMA transaction, the PIO transaction, which had been retained in the SBC, would proceed.

The PIO buffers effectively provide a single element of a write buffer, since the CPU continues to execute instructions without waiting for a PIO write to complete.

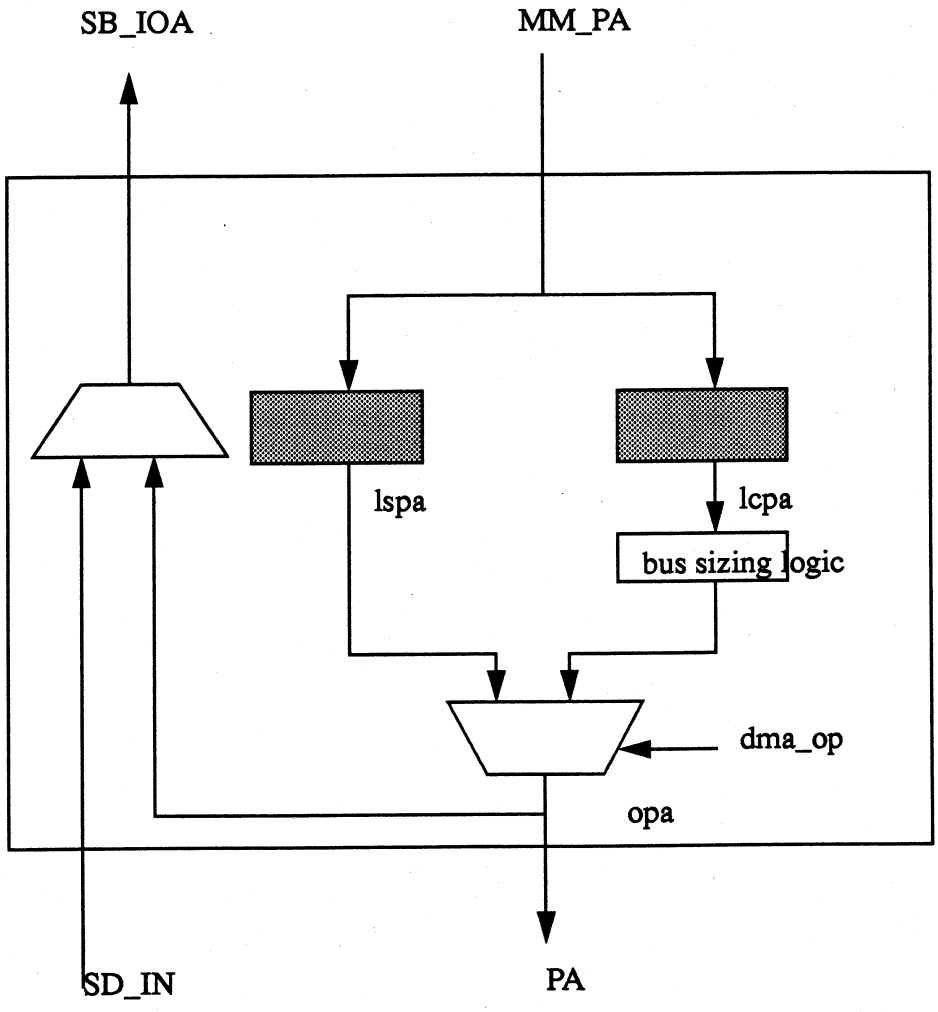
9.3 Address Control:

The Address Control Block (CAD) handles the address generation function for SBus transactions and local resources data transfers. The block insures that the proper SBus physical address is valid and stable whenever address strobe is asserted. In the case of 8 word burst write. SBC will break the cycle into 2 quad word tranfers with 2 separate IOREQ. SBC also increments the address for the second quadword transfer. There are two sources for an SBus physical address; the CPU generates a virtual address during PIO transactions and the SBus master generates a virtual address during DVMA transactions. In both cases the MMU translates the virtual address to a physical address. Since PIO transaction can start during a DVMA and vice versa, both physical addresses must be retained by the Address Control Block. The only time the CAD block manipulates the SBus physical address is during double word catabolism and dynamic bus sizing.

In order to deal with such implementation-specific processes as memory burst order and local resource transfer sizes, the CAD block manipulates

some low-order address bits to simplify data transfer control. In either case, data is transferred properly and control logic is simplified. For SBus cycle order of burst data complies with SBus P1496 spec. For MEMIF / SBC burst transfer, the order of data is the same as in microSPARC except in the case of double word burst write with PA[2] is 1 (odd word). In this case the order of data transfer will be the same as if PA[2] is 0. In the case of 4 and 8 word burst write, SBC will force address [3:2] to zero. Thus the transfer from SBC to MEMIF always start at the beginning of the block.

Figure 9.1 - AddressControl Block



9.4 SBus Arbiter:

The SBus Arbiter handles bus requests from the CPU and as many as six DVMA masters. The SBus arbiter employs all fairness, and arbitration protocol as outlined in the SBus Specification P1496.

The fairness algorithm utilizes a token, which is passed round robin style. All seven masters are given tokens which are prioritized based on the last master to have owned the bus. The requesting master with the highest priority is granted the bus. Once that master is finished with the bus, new tokens are assigned. The last owner is given the lowest priority.

The CPU is treated as one of the seven masters. In this regard, the CPU master is indistinguishable from any other DVMA master. In addition to this, there are two ways in which the CPU is given special treatment. If the bus is free and is not about to be granted, the CPU has the ability to anticipate that its request will be granted. In this fast-bus-access case, the CPU will forego waiting for the bus grant in order to begin the bus cycle.

Another special case is made during times when a PIO transaction is dynamic bus sized. An attempt is made to keep the follow-on cycles atomic with the first cycle (although a retry or error will cause the atomicity to be broken) in order to help the latency of that cycle.

9.5 Main Control:

The Main State Machine (MSM) controls the internal data store, issues data transfer and translation requests to the MMU, and generally acts to coordinate the other state machines in the SBC. One of the major functions of the SBC is to move data between local resources and SBus devices. The SBC has to fill the internal data store by a get operation and then empty the data store with a put operation. The MSM controls the above-mentioned get and put operations.

9.6 Slave Control

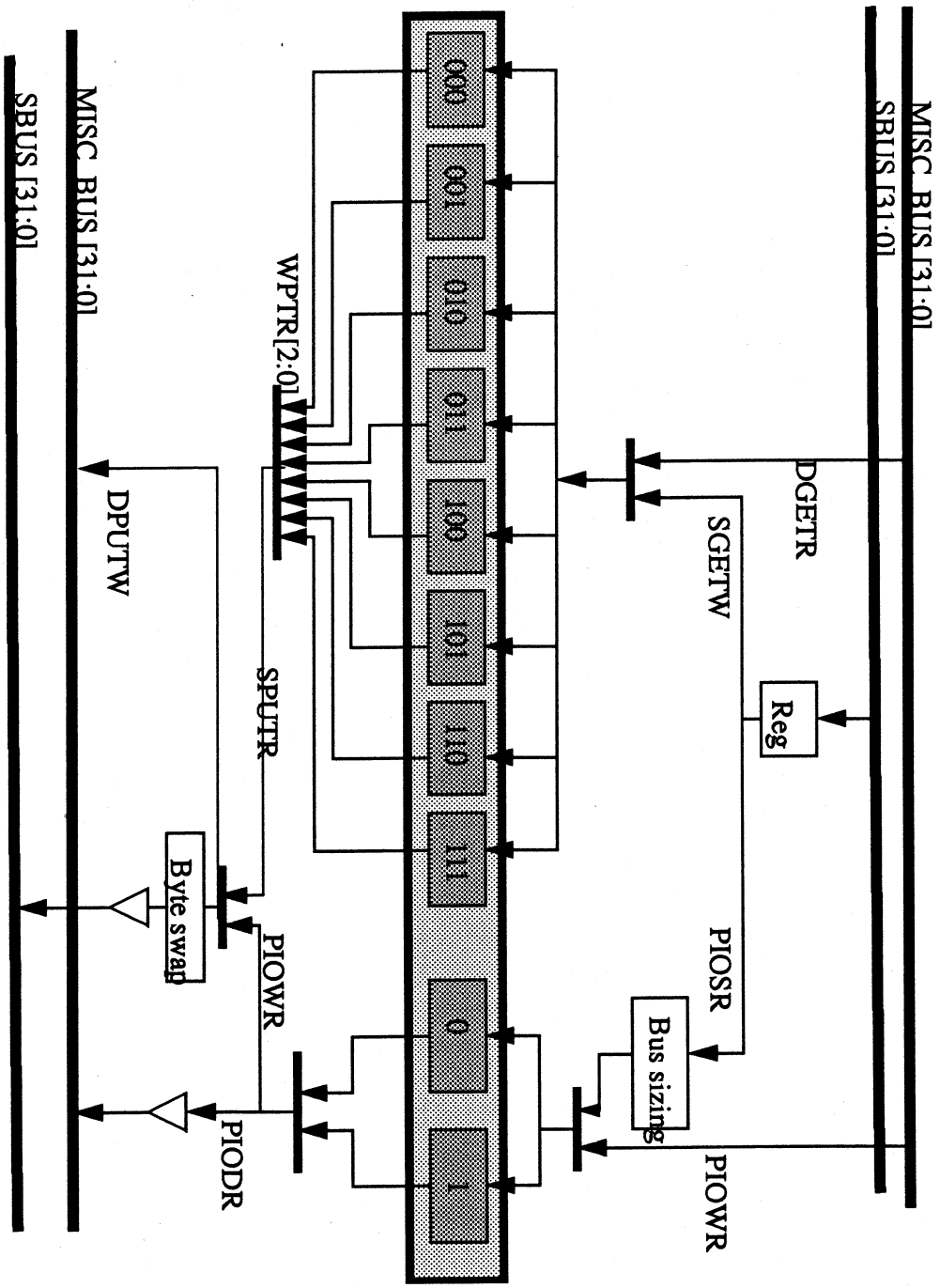
The SBus control state machine (SSM) is charged with tracking the progress of the current SBus cycle by monitoring the Transfer Acknowledgment (ACK) and terminating the cycle once completed or upon an error detection. The SSM does not differentiate between the ACKs from the TSM (Target Control State Machine) and other external ACKs; it treats the TSM as any other slave capable of responding with an ACK.

9.7 Slave Target Control

The Target control State Machine (TSM) controls the Transfer Acknowledgment (ACK) during local DVMA transactions or error conditions, when it is appropriate for the SBC to drive these signals.

9.8 Data Path

Figure 9.2 - SBC dataPath



The SBC data path consists of a series of multiplexers and registers necessary to transfer data between the SBus devices and local resources. There are two sources of data: the internal data bus, MISC BUS, which

connects local resources to the SBC and the SBus data bus. Data from the SBus is buffered, then passes through a byte swapper, which is necessary to align SBus byte or half-word ports, passes through a source select mux on its way to the internal data store. Data from the internal data bus passes through the source select mux to the internal data store.

The heart of the data path is the internal data store, which provides temporary storage for up to 40 bytes of data. DVMA has exclusive use of 32 bytes of internal storage and 8 bytes are exclusively used for PIO. Each byte-sized register corresponds to an address location. This means that data from a given address location will always be loaded into the same internal data store location, regardless of the order in which the data arrives. Data from either the internal data bus or SBus can go to either the DVMA register bank or the PIO register bank.

Data destined for the internal data bus goes from the internal data store, passes through destination select muxes, into an output buffer and is enabled onto the internal data bus by a tristate driver. Data destined for the SBus passes through destination select muxes, through an output byte swapper, necessary to support dynamic bus sizing and is enabled onto the SBus by a tristate driver.

9.9 Data Control

The SBC data path control logic steers the data through the source and destination multiplexers and loads the data into the internal data store. The source and destination multiplexers are straightforward to control. Since the get and put operations are serial, the data steering is almost static. The main state machine controls the get and put operations.

The internal data store load control works by the application of a mask to the load enables of each byte-sized register. Data can arrive in sizes of as small as a byte. As each piece of data arrives the load enable of its register is masked, thereby preserving the data until the put operation.

9.10 Error Handling

The Error Control Block handles errors that occur during PIO and DVMA transactions. There are three possible sources of error for PIO transactions. These are PIO transactions terminated by timeouts, error acknowledge, and late error. A two-bit error status field, ERR_TYPE, is used to indicate to the CPU the source of error during PIO transactions. This bus is sampled and any code other than that indicating no error

signifies that an error has occurred. During this time, the entire state of the current PIO transaction is made available to the CPU for error reporting. In case of double word split and an ERROR ACK is returned for the first word, the second word transfer is aborted and error is reported to the MMU. For PIO Read, error is reported at the same time with data availble. For bus sizing cycles, any Error ACK will abort the rest of the transfer. In the case of bus sizing late error, only one error is reported to the MMU regardless of how many late errors seen on SBus.

The sources of error for DVMA transactions are translation, parity, timeout and SBus protocol errors. Parity can occur either during address translation or a get operation from local resources. In all cases the SBC becomes the slave target and drives ACK to indicate an error to the DVMA master. Errors during DVMA are transparent to the CPU. The SBC does not use the SBus late error signal to indicate error

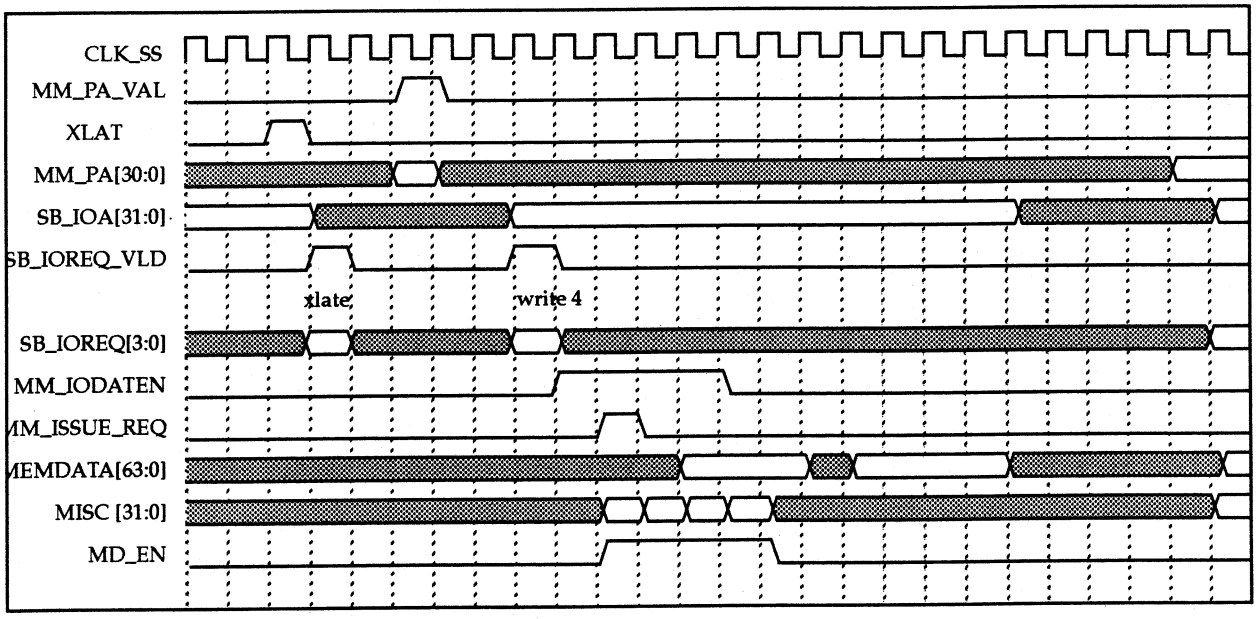
9.11 Timing Diagrams

9.11.1 DVMA Write timing

For DVMA write, there are two IOREQ's, one for address translation and one for data write
If transfer size is double word or more then address are modified to make the transfer always start at the beginning of the block The diagram below is an example of 4 word DVMA write.

PA[4:2] Order of 8 word transferred

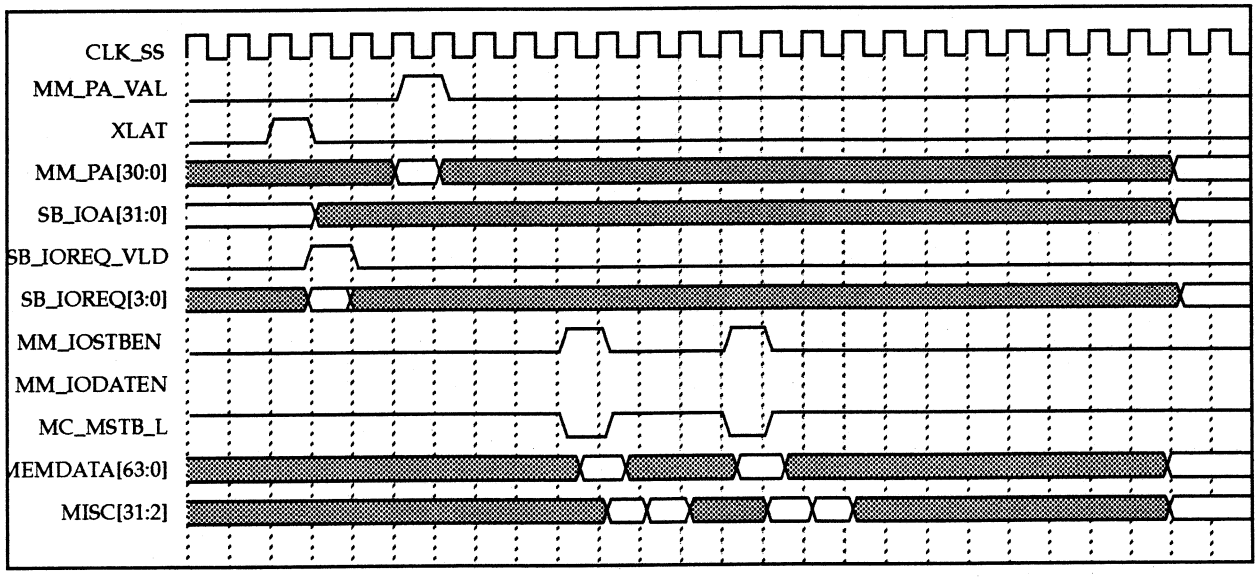
0** 0 1 2 3 4 5 6 7
1** 4 5 6 7 0 1 2 3



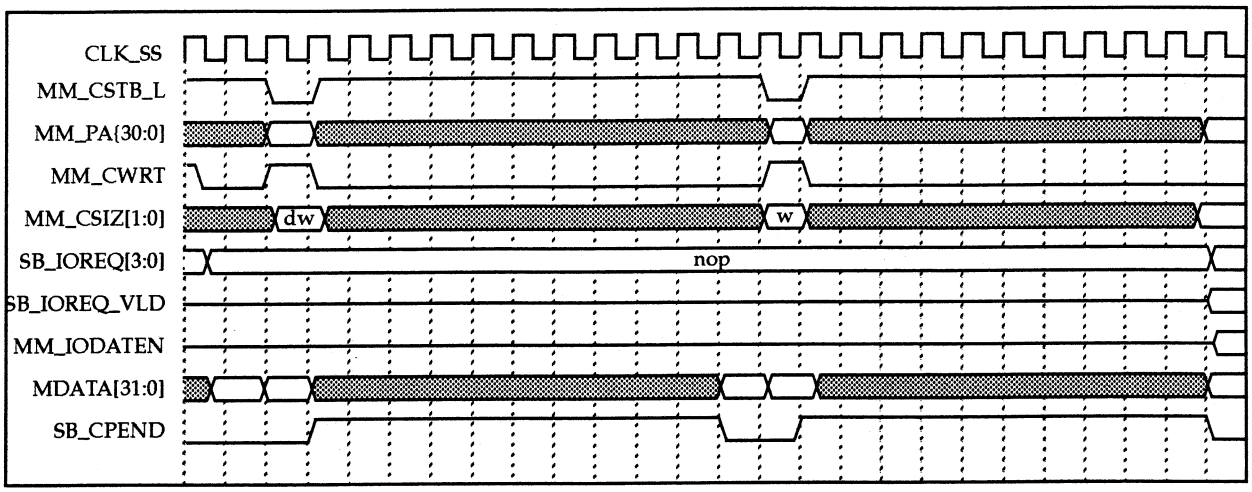
9.11.2 DVMA Read timing:

For DVMA read, there is only one IOREQ for both translation and data read. The MMU needs to decode PA[30:29] to determine the target. If the target is not memory, the MMU should not initiate the read cycles to MEMIF. If the target is memory the MMU will initiate the memory cycle depending on the size of the transfer request. The following diagram is an example of 2 word DVMA read.
Note that the order of words tranfered in a double word or more depends on the value of PA[2].

PA[4:2]	Order of 8 words transferred to
000	0 1 2 3 4 5 6 7
001	1 0 2 3 4 5 6 7
010	2 3 4 5 6 7 0 1
011	3 2 4 5 6 7 0 1
100	4 5 6 7 0 1 2 3
101	5 4 6 7 0 1 2 3
110	6 7 0 1 2 3 4 5
111	7 6 0 1 2 3 4 5

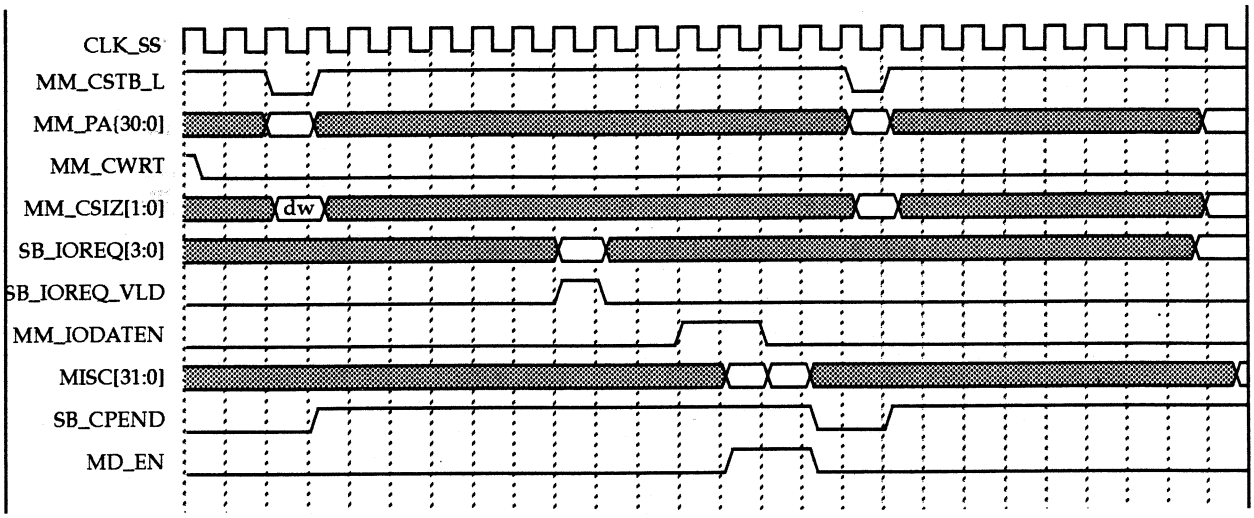


9.11.3 PIO Write timing:



The above diagram is two PIO write cycles, one double word and one single word. Note that for double word transfer, the address is always aligned.

9.11.4 PIO Read timing:



The delay from IOREQ to MM_IODATEN is approximate.

Chapter 10 Reset, Power down, PLL, Clock Control, Jtag

10.1 Overview

This section will describe the Reset logic, Phase Lock Loop, Clock Control logic and the JTAG architecture. The JTAG, reset control and clock start/stop control logic are part of the Misc block, while the clock controller and the PLL are 2 design blocks by themselves.

The JTAG logic controls all the scan operation within the chip and in conjunction with the clock start/stop logic, enables the single step operation of the chip for debug purposes. All of the registers in the chip are scannable and are configured as one single internal scan chain for testing as well as debugging the chip.

10.2 Reset Controller

When the reset input is active the microSPARC-II CPU activates the SBus reset. All RAMs including the IU and FPU register files, the data and instruction cache rams, and the TLB remain unchanged by the assertion of Reset.All other registers in microSPARC-II CPU reset to zero.

State and pipeline registers internal to the IU are established on reset via reset logic in the IU, not via explicit reset to the flip-flop. This is to support clearing and setting certain bits (e.g.: S bit of the PSR).

The microSPARC-II Reset Controller performs the simple task of driving microSPARC-II's internal reset lines, and inhibiting clocks during transitions on those lines to avoid timing violations on the flip-flops being reset.

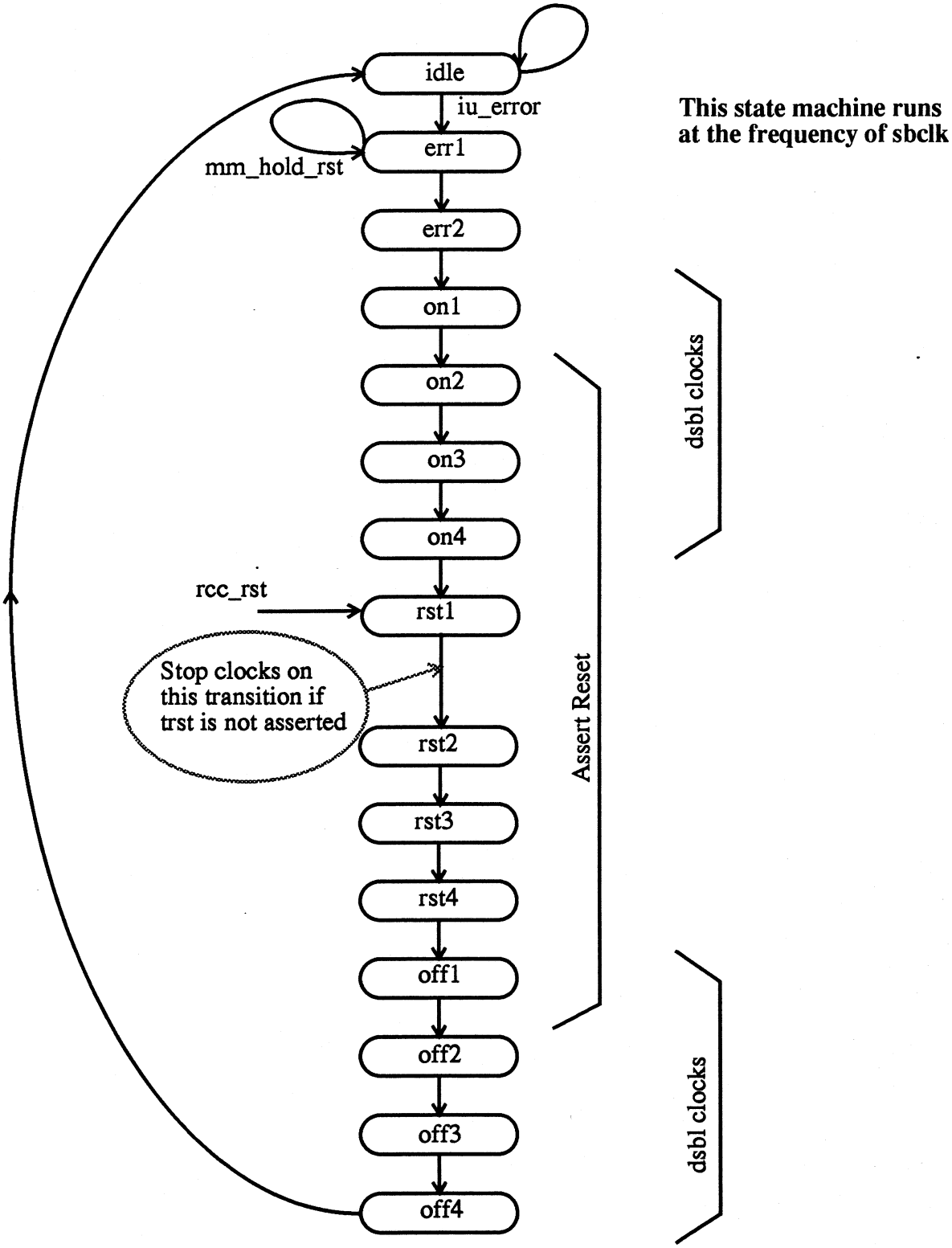
microSPARC-II has two reset operations: General Reset and Watchdog Reset. General Reset is done in response to assertion of the input_reset_l microSPARC-II input pin; this happens on powerup and on any externally-triggered reset. Watchdog Reset is performed when the IU enters error state due to taking a trap while the PSR ET bit is deasserted. General Reset will cause assertion of both Reset Controller output signals: reset_any and reset_nonwd; Watchdog Reset will cause only reset_any to be asserted. Reset_any resets the IU and any other logic which must be reset only on Watchdog Reset; reset_nonwd resets everything else except the clock and reset logic and the TAP controller.

In addition to reset_any and reset_nonwd, the reset controller has another output, rs_dsbl_clocks, which is used to disable the outputs of the clock controller during transitions on the reset lines. This allows the

heavily-loaded reset signals time to propagate throughout the chip completely between clocks, to avoid setup and hold time violations. All three of these outputs are controlled by the reset state machine. However, input_reset_l is combinationally ORed into both reset_any and reset_nonwd, and rcc_rst forces clocks to be running; taken together, these assure that any circuitry which must (for physical reasons) see reset asserted immediately on powerup will see it (assuming that input_reset_l is asserted, and input_clock is oscillating, immediately on powerup). As a consequence, timing violations may occur on the first clock after assertion of input_reset_l; the ensuing General Reset will eventually clean up any illegal states caused by these violations.

Inputs which affect operation of the reset state machine are: rec_rst, a sbclk synchronized version of microSPARC-II's input_reset_l pin; iu_error, the error state indication from the IU which initiates a Watchdog Reset; and mm_hold_rst, a signal from the MMU which delays the start of a Watchdog Reset sequence until all there are no loads, stores, or instruction fetches in progress. Rcc_rst is inhibited during scan shift operations, to prevent loss of non-resettable state if input_reset_l should happen to be asserted during a scan shift.

Figure 10.0 - Reset State Machine



10.3 Reset Controller State Machine Operation

The reset state machine is clocked at sbclk. Assertion of rcc_rst synchronously resets the state machine into the rst1 state from any other state. The state machine will thus stay in state rst1 for as long as rcc_rst is asserted. After completing a reset sequence, the state machine hangs in the idle state until either iu_error or rcc_rst is asserted. If iu_error is asserted while in the idle state, the state machine goes to state err1, waits there until mm_hold_rst is deasserted, and then completes the reset sequence and returns to idle. Reset_any and/or reset_nonwd are asserted in states on2, on3, on4, rst1, rst2, rst3, rst4, and off1: if the reset sequence was initiated by iu_error, only reset_any is asserted; if initiated by rcc_rst, both reset_any and reset_nonwd are asserted. Clocks are disabled in states on1, on2, on3, and on4 as the reset signal is turned on; they are disabled again in states off1, off2, off3, and off4 as reset is turned off again. This clock disabling does not put the clock state machine into the stopped state; it merely gates off the clock outputs. Note that the reset lines are always deasserted during a clocks-disabled period, and, for Watchdog Reset, they are asserted during a clocks-disabled period.

To facilitate scan-based debugging, the reset state machine will assert rs_stop_even upon exiting the rst1 state during a General Reset sequence. If microSPARC-II's jtag_trst_l input is deasserted at that time, this will cause the clock control state machine to enter the stopped state. The reset sequence will continue as clocks are issued under scan control. It is thus possible to single-step through the remaining states of the reset state machine, and, more importantly, to reset the machine to a known, deterministic state during scan-based debug.

10.4 Clocking and Phase lock loop requriement

microSPARC-II uses a Phase Locked Loop design to generate the internal high frequency clock. The following is the PLL block diagram.

The ss_clock is the system clock inside microSPARC-II, targeted to be at 70Mhz. The sbclk is the clock for the SBC. This is generated by dividing the ss_clock by 3 , 4 or 5 to generate a clock frequency between 16.6Mz to 25 Mhz as required by the SBus specification. The ref_clk has the same frequency as the ss_clock and is sent off chip for testing purpose.The gclk is the graphics clock for the Local Graphics bus. This is generated by dividing the ss_clock by 3.

The voltage control oscillator (VCO) will generate a clock with frequency 2x of the ss_clock. (140Mhz). In normal operation, the frequency of ext_clk1 is 1/2 of the ss_clock frequency. Ext_clk2 is tied to ground. An independent path is provided to bypass the PLL. When

this path is used, ext_clk1 and ext_clk2 will be 90 degree out of phase. They are being XORed to generate 2x the frequency. PLL_BYP_L is used to select the clock controller input from either the PLL or the bypass path. DIV_CNTL_ is used to select the divider ratio for the sb_clk (3 , 4, or 5).

The following expression summarizes the clock generation:

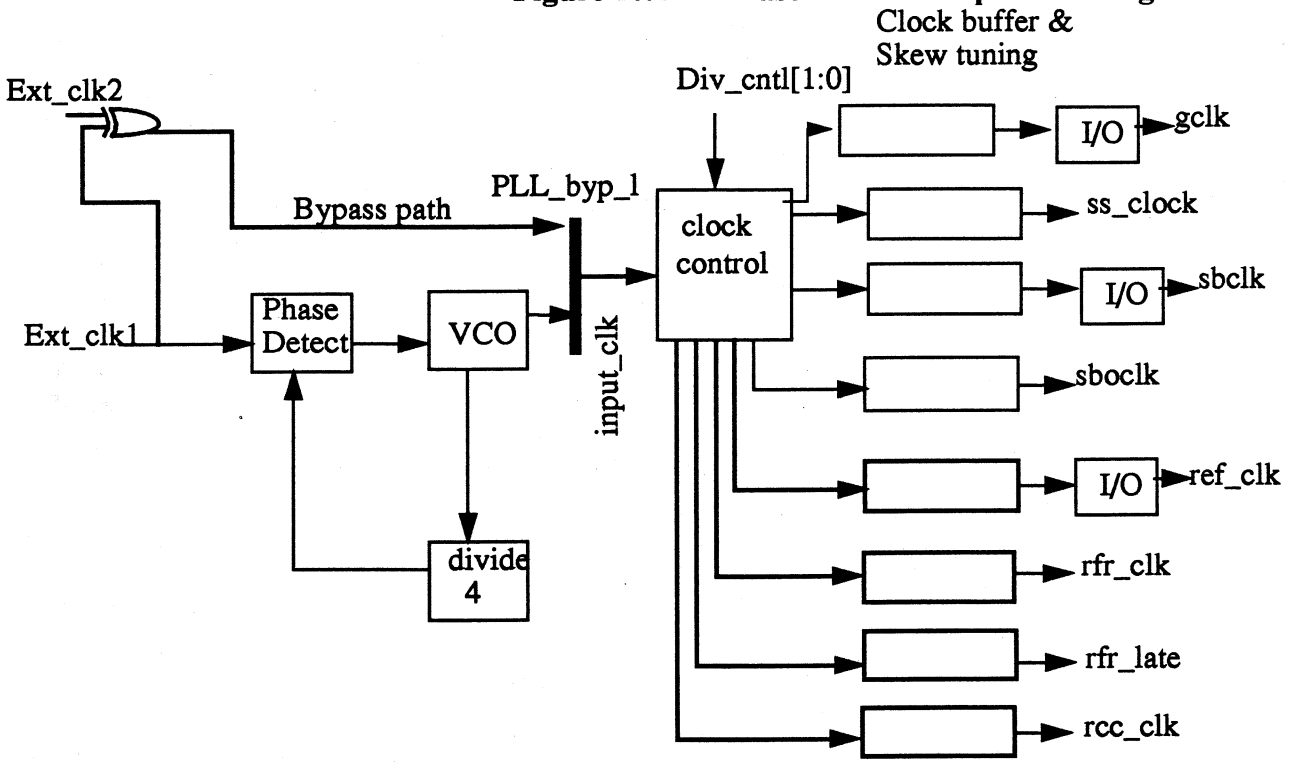
input_clk = PLL_byp_l?

(4x ext_clk1 frequency):(ext_clk1 XOR ext_clk2)

ss_clock will be half the frequency of input_clk.

Clock skew between ss_clock and sb_clk , ss_clock and ref_clk should be less than 1 ns. The PLL is designed to operate up to 125Mhz.

Figure 10.1 - Phase Locked Loop Block DIagram.



10.5 Power Management

List of microSPARC-II power management features:

- Cache RAM powerdown: whenever the cache controllers detect that one of the cache RAMs need not be accessed in a given clock cycle, that RAM is automatically put into 'powerdown' mode for that cycle. In this mode the RAM consumes minimal power. This mode

is used when the cache is disabled, when the CPU is waiting for cache miss data to be returned from memory, and when the chip is in 'standby' mode (see below).

- Standby mode. In this mode, internal clocks are inhibited to all logic except the DRAM refresh controller and the clock controller. This mode is controlled by the 'standby' input pin on the microSPARC-II chip. When the standby pin is asserted, internal logic waits until the chip is in a 'safe' state (i.e. no pending memory operations or SBus transfers), then turns off the internal clocks. Clocks stay off until the standby pin is de-asserted. The cache RAMs are put into 'powerdown' mode (see above) while the chip is in standby mode. In standby mode, sbclk continues to oscillate, but the SBC will not respond to any requests; DRAM refresh continues as normal. Latency from de-assertion of the standby pin to resumption of internal clocking is on the order of a few sbclk periods. The standby input signal must meet setup and hold requirements to sbclk, but otherwise is unconstrained.
- Self-refresh DRAM mode. In this mode, the DRAMs operate in self-refresh mode (assuming that the DRAMs have self-refresh capability). It is controlled by bit 13 of the PCR. After PCR[13] is written to 1, the DRAMs will enter self-refresh mode within 2 microseconds.

10.6 Clock Controller

The microSPARC-II Clock Controller generates the clock signals used by all of microSPARC-II (except the TAP controller), as well as the sbclk used by external SBus interface devices. Its operation is controlled by the Clock Control Register (CCR), a collection of internal register bits which are writeable only by scan. On Reset, the CCR is cleared. Subsequent scan shift operations can be used to set bits of the CCR in order to alter the operation of clock state machine, as described below.

The microSPARC-II clock controller is designed to interface to a simple internal cycle counter (ICC) for precise, at-speed control of system clocking. The ICC is a simple binary counter, accessible by board-level scan, which increments on sbclk positive edges. The interface consists of three microSPARC-II I/O pins:

* sbclk (output) - the SBus clock output, which is gated off when system clocks are turned off. This output is used to clock the external SBus logic as well as theICC.

* ext_event (input) - this input is immediately registered in a sbclk-clocked flipflop. Under control of some Clock Control Register (CCR) bits (which are writeable only by scan), a logic 1 in this flipflop will cause clocks to stop either at the next rcc_clk edge or the next sbclk edge. This input should be driven by the terminal_count output of the ICC, perhaps ORed with other externally-detected clock stop signals. In a standard binary up-counter, the terminal count output is asserted when the counter contains all 1's (i.e. a two's-complement value of -1).

* int_event (output) - this is the output of a sbclk-clocked flipflop. It is asserted whenever an internally-detected 'event' occurs (e.g. virtual address match). These events can, under control of some CCR bits, stop clocks; however, whether or not they stop clocks, they always cause assertion of the int_event output. This output can be used to trigger a logic analyzer; in addition, it can be used in conjunction with the ICC as described below to implement the 'stop N cycles after internal event' function.

In addition, there are two microSPARC-II input pins which control the internal clock divider: it specifies the (rcc_clk:sbclk) frequency ratio, D:

* div_ctl[1:0] (input)

div_ctl [1:0]	D	rcc_clk range Mhz	phi[2:0]
00	2	33.3 - 50	0,1
01	3	50 - 75	0,1,2
10	4	66.7 - 100	0,1,2,3
11	5	83.3 - 125	0,1,2,3,4

The 'rcc_clk range' shown above is the range of internal rcc_clk frequencies that is obtained when sbclk spans its legal range of 16.7 - 25 MHz. The 'phi[2:0]' column shows the sequence of states traversed by the phi[2:0] field of the CCR in each sbclk cycle: phi[2:0] transitions to the next state in the sequence on each rcc_clk rising edge, and the rcc_clk rising edge which coincides with the sbclk rising edge always causes phi[2:0] to transition to the 0 state.

Note that the ICC/microSPARC-II interface runs at the SBus clock rate, and the signal I/O connect directly to inputs or outputs of flipflops within microSPARC-II; thus, the ICC logic has nearly a full SBus cycle in which to set up its output to the ext_event input.

10.6.1 Stopping Clocks

This does not require the use of the ICC. To stop clocks, set the stop_clocks CCR bit.

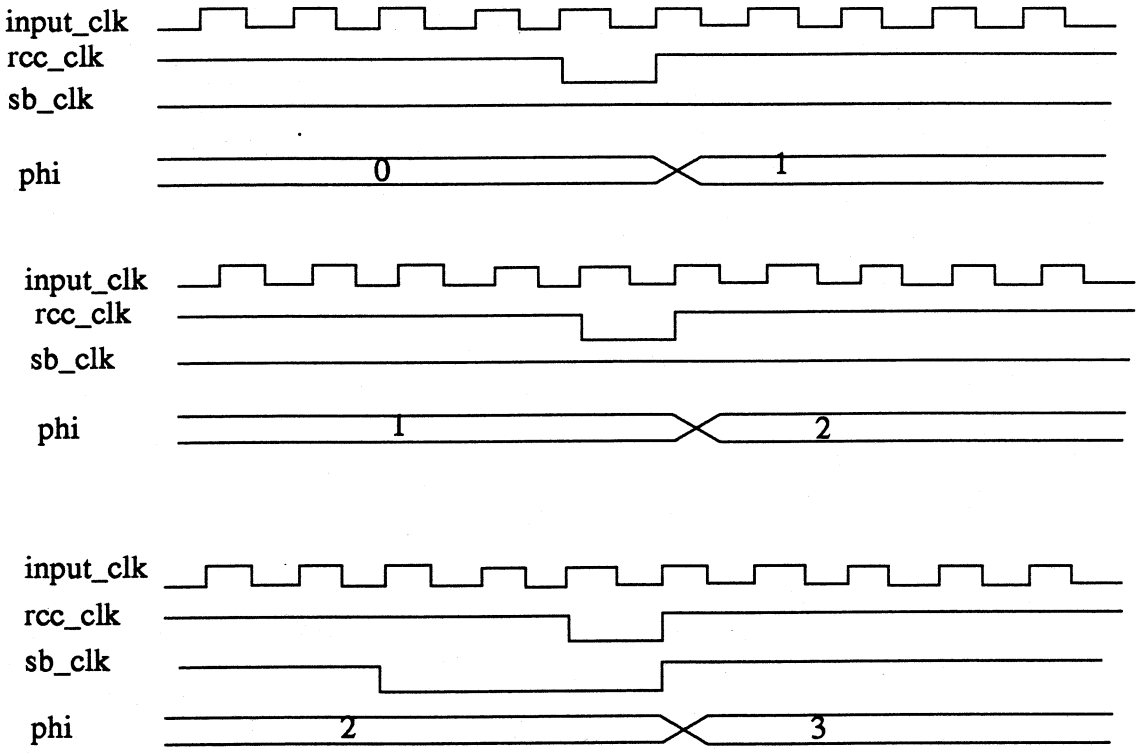
10.6.2 Starting Clocks

This does not require the use of the ICC. To start clocks, set the start CCR bit.

10.6.3 Single-Step

This does not require the use of the ICC. From a clock-stopped state, set both the stop_clocks and start bits of the CCR. A single active-low rcc_clk pulse will be issued, with a pulse width of 1/2 the normal rcc_clk period; if the rcc_clk pulse causes phi[2:0] to transition to the 0 state, a single active-low sbclk pulse will also be issued (its pulse width is 1/2 the normal sbclk period, and its rising edge will coincide with the rising edge of rcc_clk

Figure 10.2 - divide-by-3 example:



10.6.4 Counting Clocks

When the ICC is enabled, it increments on every sbclk positive edge. Since the states of the ICC and the CCR are accessible via scan, the number of clocks issued between any two points in time can be calculated by scanning out this state information before clocks are

started and again after they have been stopped. The following formula can be used.

N = D*(ICC.after-ICC.before) + (phi.after-phi.before)

D is the divider ratio (2, 3, 4, or 5) specified by div_ctl[1:0]; ICC.before and ICC.after are the respective values of the external clock counter before and after clocks have been issued; phi.before and phi.after are the corresponding values of the phi[2:0] bits of the CCR.

The formula above of course assumes that ICC has not wrapped around; the ICC control logic should contain a wraparound detector that can be read by scan.

10.6.5 Issuing N Clocks

The ICC can be used to issue exactly N system clocks, at full speed. N can be any number from 1 to approximately D*(2^X), where D is the (rcc_clk:sbclk) frequency ratio and X is the number of bits in ICC; for example, a 32-bit ICC lets us control clocks over a 200-second range at 80-MHz operation in divide-by-4 mode. This function does not require the use of the int_event output.

To issue N clocks from a clocks-stopped state, several CCR fields, as well as the ICC register, are involved. You must scan a 1 into the start and stop_on_ext_event control bits, copy (using scan) the current phi[2:0] field into the ref_phi[2:0] field, and scan appropriate values into the extra_cycles[2:0] field and into the ICC. The number of clocks issued is given by this formula:

N = D*(-ICC.before) + extra_cycles + 1 ;

where -ICC.before is the positive number gotten by taking the twos-complement of the scanned-in ICC value. Thus, to issue N clocks, scan the twos-complement of (N-1)/D into the ICC, and scan (N-1)%D into extra_cycles[2:0], where '/' is integer divide with the remainder discarded, and '%' is the remainder of integer divide. For example, to issue 17 clocks in divide-by-3 mode, you would scan -((17-1)/3) = 0xfffffffb into the ICC, and (17-1)%3 = 1 into extra_cycles[2:0] .

Because the value scanned into the ICC is treated as a negative number to be counted up towards zero, the formula above works only when (N-1)/D > 0, i.e. when (N > D). For (0 < N <= D), scan 00000000 into the ICC, scan 1 into the ext_event_sb1 bit of the CCR, and scan (N-1)%D into extra_cycles[2:0] .

Here's a complete algorithm, including a few other CCR bits which must be set to specific states:

```
if (N < 1) error ;
else {
   ICC = -(N-1)/D ;
   CCR.extra_cycles = (N-1)%D ;
   CCR.ref_phi = CCR.phi ;
   if (N <= D) CCR.ext_event_sb1 = 1 ;
   else      CCR.ext_event_sb1 = 0 ;
   CCR.start = 1 ; CCR.stop_on_ext_event = 1 ;
   CCR.stop_int_to_ext = 0 ;
   CCR.int_to_ext = 0 ;
   CCR.ext_event_sb2 = 0 ;
     }
```

10.6.6 Stop Clocks on Internal Event

This does not require the use of the ICC. To stop clocks on detection of an internal event, set the stop_on_int_event bit of the CCR and enable the desired internal event detection logic. Clocks will, with some limitations, stop at the end of the rcc_clk cycle in which the input to the int_event flipflop is asserted. The limitation of this mode is that clocks cannot stop in phi==2 when D==3, phi==3 when D==4, or in phi==3 or 4 when D==5; if an internal event occurs in either of these situations, clocks will stop one cycle later, in phi==0. Note that, since the int_event flipflop is clocked only on sbclk edges, the int_event output pin will not be set by the internal event which stops the clocks, unless clocks have stopped in phi==0.

10.6.7 Stop Clocks N Cycles after Internal Event

In this mode, the ICC is held until an internal event occurs. The internal event does not stop clocks, but does cause assertion of the int_event output; the int_event output will remain asserted until it is cleared by scan. The ICC is enabled to count whenever int_event is asserted, so clocks will continue to run until ext_event is asserted, either by ICC or by another external event detector. The intent of this mode is to issue exactly N more clocks than would have been issued in stop_on_int_event mode (see above); i.e. exactly N clocks will be issued **after** the first rcc_clock positive edge at which the input to the int_event flipflop is asserted. Logic in the clock controller records the clock phase in which the internal event occurred, and this information is factored

into the subsequent clock stop on external event, so that N can be any integer. Due to timing limitations, N must be greater than D.

To support this mode, the ICC must have logic which, under scan control, holds the count when int_event is not asserted.

To have clocks continue for exactly N cycles after the cycle in which the internal event occurs, several CCR fields, as well as the ICC register, are involved. You must scan a 1 into the start and int_to_ext CCR bits, scan a 0 into the stop_on_ext_event and stop_int_to_ext CCR bits, and scan appropriate values into the extra_cycles[2:0] field and into the ICC. The following formula gives the number of addtional clocks to be issued after the cycle in which the internal event occurs:

```
N = D*(-ICC.before) + extra_cycles + D ;
```

where -ICC.before is the positive number gotten by taking the twos-complement of the scanned-in ICC value. Thus, to issue N clocks, scan the twos-complement of (N/D - 1) into the ICC, and scan (N%D) into extra_cycles[2:0], where '/' is integer divide with the remainder discarded, and '%' is the remainder of integer divide. For example, to issue 35 clocks after an internal event in divide-by-4 mode, you would scan -(35/4 - 1) = 0xfffffff9 into the ICC, and (35%4) = 3 into extra_cycles[2:0] . As described above for stop_on_ext_event mode, if the formula gives an initial ICC value of 0, you must also scan a 1 into ext_event_sb1 . Here's a complete algorithm:

```
if (N <= D) error ;
 else {
 ICC = -(N/D - 1) ;
 CCR.extra_cycles = (N%D) ;
 CCR.ref_phi = CCR.phi ;
 CCR.start = 1 ; CCR.int_to_ext = 1 ;
 CCR.stop_on_ext_event = 0 ;
 CCR.stop_int_to_ext = 0 ;
 CCR.int_event = 0 ;
 if (N < (2*D)) CCR.ext_event_sb1 = 1 ;
 else           CCR.ext_event_sb1 = 0 ;
 CCR.ext_event_sb2 = 0 ;
```

```
}
```

10.6.8 Stop Clocks after N Internal Events -

In this mode clocks are stopped after the Nth detected internal event. Clocks are stopped as described above for stop_on_int_event mode (see above), except that the first (N-1) sbclk cycles of int_event assertion are ignored. Due to the limited resolution of the ICC interface, if more than one internal events occurs within a single sbclk cycle, that counts as only a single event.

This mode is enabled by the stop_nth_event CCR bit, and ICC needs a scannable control bit which enables it to count only while int_event is active.

To use this mode, you must load ICC with (2-N) and turn on stop_nth_event. As with other modes described above, some special action is required if the initial ICC value given by this formula is non-negative. Here is a complete algorithm:

```
if (N < 1) error ;
else {
ICC = (2 - N) ;
    CCR.start = 1 ;
    CCR.int_to_ext = 0 ;
    CCR.stop_on_ext_event = 1 ;
    CCR.stop_int_to_ext = 0 ;
    CCR.int_event = 0 ; if (N == 1)
     CCR.ext_event_sb2 = 1 ; else
     CCR.ext_event_sb2 = 0 ;
    if (N == 2)  CCR.ext_event_sb1 = 1 ;
    else        CCR.ext_event_sb1 = 0 ;
      }
```

10.6.9 CCR Bits

Here is a list of the Clock Control Register bits. These are accessible by scan only, and their functionality is described above.

start

stop_clocks

stop_on_int_event

stop_on_ext_event

stop_int_to_ext

stop_nth_event

extra_cycles[2:0]

int_event

ext_event_sb1

ext_event_sb2

phi[2:0] (Treat this as read only)

ref_phi[2:0]

10.7 JTAG

A variety of microSPARC-II test and diagnostic functions, including internal scan, boundary scan and clock control, are controlled through an IEEE 1149.1 (JTAG) Standard Test Access Port (TAP). Commands and data are sent as serial data between the JTAG master and the microSPARC-II chip (a JTAG slave), via a 4 wire serial testability bus (JTAG bus). The TAP interfaces to the JTAG bus via 5 dedicated pins on the microSPARC-II chip. These pins are:

TCK - input - test clock

TMS - input - test mode select

TDI - input - test data input

TRST_L - input - JTAG TAP reset (asynchronous)

TDO - output - test data output

For more details on the IEEE protocol, please refer to the IEEE document "IEEE Standard Test Access Port and Boundary-Scan Architecture", published by IEEE.

10.8 Board Level Architecture

Any microSPARC-II based system will contain several JTAG compatible chips. These are connected using the minimum (single TMS signal) configuration as described in the 1149.1 specification (Figure 3-1, IEEE 1149.1 standards manual). This configuration contains three broadcast signals (TMS, TCK, and TRST,) which are fed from the

JTAG master to all JTAG slaves in parallel, and a serial path formed by a daisy-chain connection of the serial test data pins (TDI and TDO) of all slaves.

The TAP supports a BYPASS instruction which places a minimum shift path (1 bit) between the chip's TDI and TDO pins. This allows efficient access to any single chip in the daisy-chain without board-level muxing.

10.9 TAP

The TAP consists of a TAP controller, plus a number of shift registers including an instruction register (IR) and multiple "data" registers.

The TAP controller is a synchronous FSM which controls the sequence of operations of the JTAG test circuitry, in response to changes at the JTAG bus. (Specifically, in response to changes at the TMS input with respect to the TCK input.). Note that the TAP controller is asynchronous with respect to the system clock(s), and can therefore be used to control the clock control logic. The TAP FSM implements the state (16 states) diagram as detailed in the 1149.1 protocol.

The IR is a 6-bit register which allows a test instruction to be shifted into microSPARC-II. The instruction is used to select the test to be performed and/or the test data register to be accessed. The supported instructions are listed in a later section.

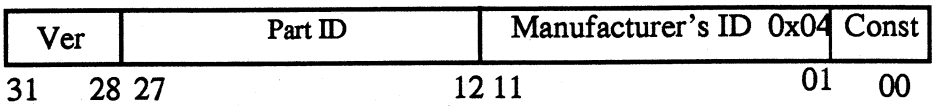
10.10 Data Registers

Although any number of loops may be supported by the TAP, the FSM in the TAP controller only distinguishes between the IR and a data register. The specific data register is decoded from the instruction in the IR.

The following data registers are supported in the microSPARC-II TAP:

- Bypass Register - a single bit shift register for efficient board-level scan.
- Device I.D. Register - a 32-bit register with the following field.

Figure 10.3 - JTAG ID Reg Contents



Field Definitions:

Version - Bits[31:28} represent the version number which is 0x0 for this version

Part ID - Bits[27:12] represent part number as assigned by Fujitsu, which is 0x0000

Miff ID - Bits[11:01] represent manufacturer's ID as per JEDEC, which is 0x04

Const - Bit[0] is tied to a constant logic'1'

Value in ID Register: 32'h 00000009

- Data registers - A two bit clock control register to sample outputs from the clock controller(CCR)
- Boundary Scan Register - a single scan chain consisting of all of the boundary scan cells (input, output and inout cells).
- Internal Scan Registers - a single scan chain of all the internal scan flipflops

10.11 JTAG Instructions

The following instruction are supported by the microSPARC-II TAP. The table contains the bit-value and mnemonic, as well as which data register is selected by that instruction. The encodings followed by an "*" are fixed by the IEEE JTAG protocol.

Table 49 - JTAG INSTRUCTIONS

value	Name of Instrn	Data register(s)	Scan Chains Accessed
000000*	EXTEST	Boundary Scan Register	Boundary Scan Chain
000001*	SAMPLE	Boundary Scan Register	Boundary Scan Chain
000010	INTEST	Boundary Scan Register	Boundary Scan Chain
000011	ATEINTEST	Boundary Scan Register	Boundary Scan Chain
100000	IDCODE	JTAG ID Register	ID Register Scan Chain
111111 *	BYPASS	Bypass Register	Bypass Register
011110	SEL_CCR	Clock Control Register	Clock Control Register Chain
010000	SEL_INT_SCAN	Internal Scan Register	Internal Scan Chain
011111	SEL_DBG_SCAN	Internal Scan Register	Internal Scan Chain
100000	CLK_RST	IBypass Register	Bypass Register

Note: 1. The two internal scan chain instructions differ with respect to the scan chain clocking during CAPTURE_DR state of the TAP fsm. Sel_int_scan will be used for ATPG tests, where a clock pulse is needed to capture the next state when scan_mode signal is in the inactive state between shift cycles. The other scan instruction, Sel_dbg_scan is used during debug to read and write the scan chain. No pulse is generated during the transition from "shift --> capture ---> shift" states. In other words, the scan state is preserved during the shift, capture, shift cycle.

2. The TDO output becomes valid at the falling edge of TCK, per the 1149.1 protocol. This is so, that the TDI input (which is connected TDO of the preceding component) of the component is stable to be clocked in during the rising edge of TCK.

3. The ATEINTEST operation is used to load the boundary scan flipflops after which, if it enters the 'run_test_idle' state, the JTAG controller will generate a single TCK pulse.

Although, the capability exists to single step the chip thru another mechanism (using sys_clock itself), ATEINTEST option provides the capability to perform ICT on the ATE, perhaps at slow speed.

4. The INTEST operation has been added so that it can be used in conjunction with the SEL_INT_SCAN instruction to perform the ATPG test using scan tool. This instruction will not generate any extra clock pulse in run_test_idle state. This is used primarily to load the boundary scan chain.

5. The Sel_CCR is used to sample two bits (stopped, sbus_1st_half) from the clock controller block. These two bits are synchronized (2 stage synchronizer using TCK) before being sampled during the shift-DR state.

10.12JTAG Interface to MISC

The JTAG block provides two key signals to the clock controller section, two signals directly to the microSPARC-II core and a five wire control signal to the boundary scan flipflops.

Clock Controller Interface:

Testclk and Testclken are the two signal that are generated in the JTAG block and sent to the clock controller.

Testclken is an active high signal that switches the ss_clock (the 70MHz) to the core from the normal 70MHz clock to the Testclk.This happens only for certan JTAG instructions. They are:

sel_int_scan, sel_dbg_scan, intest, ateintest

For all other instructions (extest, sample, bypass, idcode, sel_ccr) testclken remains inactive thus enabling the normal 70 MHz clock to microSPARC-II core. The Testclken signal is synchronized inside the clock controller using the SBus clock. By design Testclken is generated to be active at least three TCK cycles before the Testclk signal becomes active. Testclken signal changes state only during the transition from exit1-IR state of the instruction scan cycle.

Testclk is a gated version of TCK and the gating signals are sel_instruction and shift (function of shift_DR) and capture (capture-DR) states.

microSPARC-II Core Interface:

Sys_sen (ss_scan_mode) and tg_strobe are two signals that go directly to the core of microSPARC-II. Scan_mode signal is active high

whenever the TAP enters any of the four DR states: shift, exit1, pause or exit2. During the last three states, Testclk will not toggle and the state of the flipflop remains the same as the last bit scanned in during the shift state. It is necessary to activate the scan_mode signal during these three states, so that tri-states would remain disabled during repeat scan after going thru exit1, pause, exit2 states. Sys_sen is a registered signal that is clocked on the falling TCK. This has been done to avoid race conditions between the scan_mode signal and the shift clock(testclk) during the shortest tap state traversal from select-DR to shift-DR.

Since the Sys_sen is a heavily loaded (goes to all flipflops in the chip) signal, it may have a longer rise time and not meet the setup time requirement for the shortest tap state traversal to from select-DR to shift-DR. In such a case, the TCK should not be run at greater than 5 MHZ.

The tg_strobe signal is a pulse that is used as a self-timing trigger for the megacells . It is generated during the update-DR state and adheres to the timing specified in the megacell document.

Boundary Control Interface:

The five wire boundary control signal corresponds to: bin_cap, bout_cap, b_sen, b_uen, b_mode.

bin_cap and bout_cap are generated during the capture-DR state and are used to load the value on the pins or the output of the core to the boundary scan flipflop. b_sen is generated on the falling edge of the tck (to avoid race conditions) and is used as a scan_en signal for the boundary scan flipflop. b_uen is an update signal for the boundary scan update latch and it happens at the falling edge of tck.

b_mode is a mux control signal that selects between the direct pin input and the value in the update latch. This signal will change during the update-IR state and when the tap goes back to test-logic-reset state on the falling edge of TCK.

RESET Mechanism:

There is also an independent TRST_L signal which when active low would set the TAP into the tap_logic_reset state. This signal will asynchronously set the tap state machine to the tap_logic_reset state. It adheres to the 1149.1 IEEE protocol with respect to the initialization thru reset mechanism. There is no minimum active time requirement on this reset signal. If the board is not going to have an extra oscillator

for TCK, then the JTAG reset pin (TRST_L) can be tied to an active low signal thus disabling JTAG operations in the chip.

The TDI and TMS inputs have pullups on the pad and when left unconnected will be equivalent to a signal value '1' on these pins. With a free running TCK, it would guarantee that the TAP would get into the tap_logic_reset state at the end of five TCKs.

10.13JTAG Operation

The following are some of the basic operations which, when combined together will enable the user to run any of the JTAG instructions specified above. They are provided here just for understanding the TAP state transitions during various JTAG operations.

The JTAG I/O consists of TCK, TMS, TDI, TRST and TDO. The first four are inputs and the last one is the output. All five are chip I/O. The other inputs to the chip are either in a don't care state or in a predetermined state. They should not affect the operation of the JTAG controller. It should be noted, that, for a more robust operation of the chip, a proper procedure should be followed with regard to getting in and out and back to JTAG operations.(for instance resetting the system before and after JTAG operations. Once in the tap_logic_reset state, all outputs from JTAG become inactive and the chip should be back to normal functional mode.)

The tap state encodings (in hex) are as follows:

f-*test-logic-reset*, c-*run-test-idle*, 7-*select-DR*, 6-*capture-DR*, 2-*shift-DR*, 1-*exit1-DR*, 3-*pause-DR*, 0-*exit2-DR*, 5-*update-DR*, 4-*select-IR*, e-*capture-IR*, a-*shift-IR*, 9-*exit1-IR*, b-*pause-IR*, 8-*exit2-DR*, d-*update-IR*.

In order to run the JTAG instructions, the following tap state traversal is done for the various sub tasks:

Instruction Scan:

f -->c -->7 -->4 --> e -->9 -->b -->8 -->a (for 6 clocks) --> 9

(the opcode is shifted thru tdi while in the shift-IR state)

Data Scan:

9 -->b --> 8 --> d --> c -->7 -->6 -->1 --> 3 -->0 -->2(# of

shifts equal to length of scan chain) --> 1

(At state 'd' the decode instruction is latched on the falling edge of tck. Data is shifted into appropriate data register during shift cycle and at the end of shift exit to exit1-DR(1) state.

Return to new instruction:

2 --> 1 -->3 --> 0 --> 5 --> c

(wait in state c (run-test-idle) and go back to instruction scan as shown above.)

Figure 10.4 - JTAG LOGIC BLOCK DIAGRAM

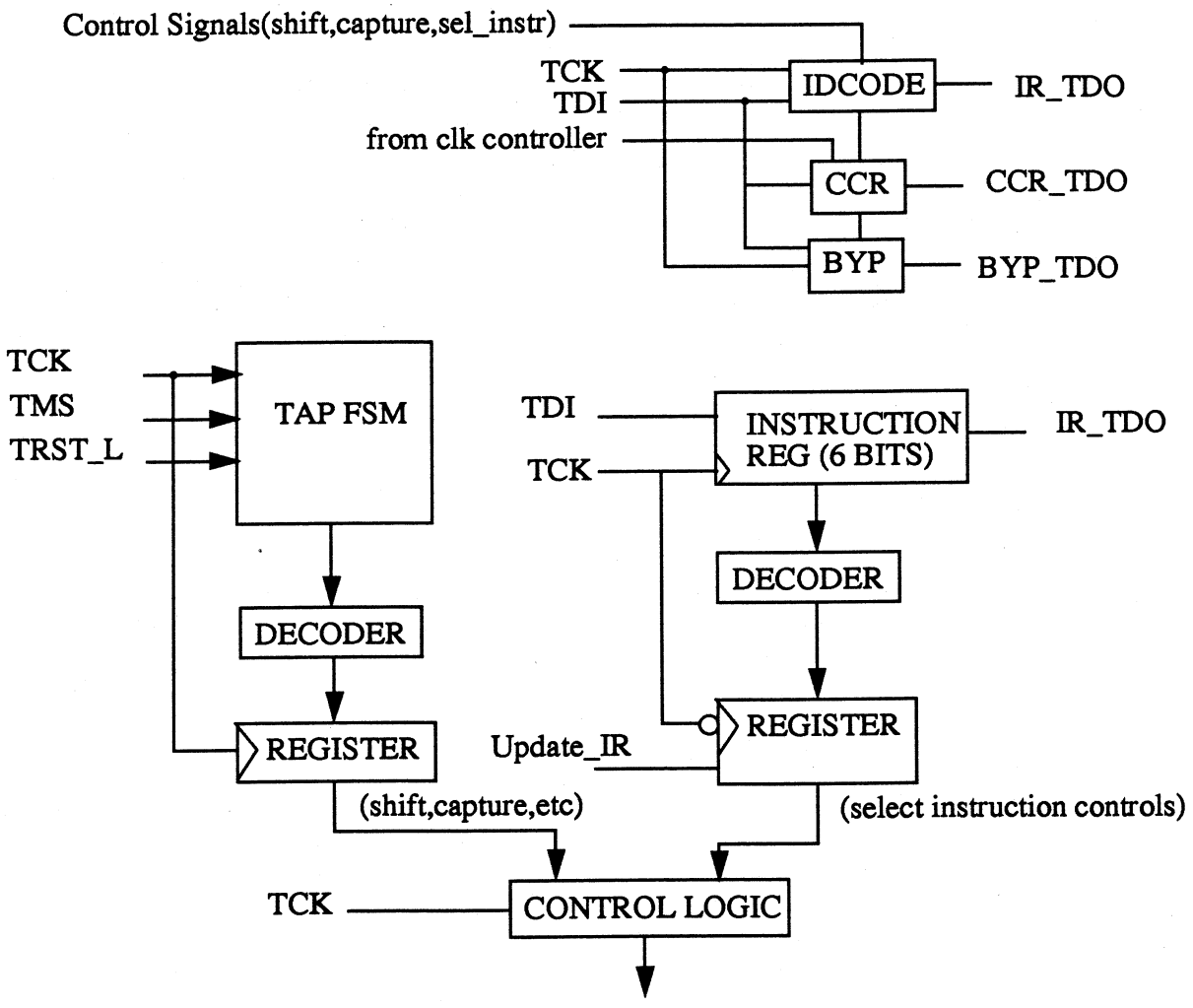
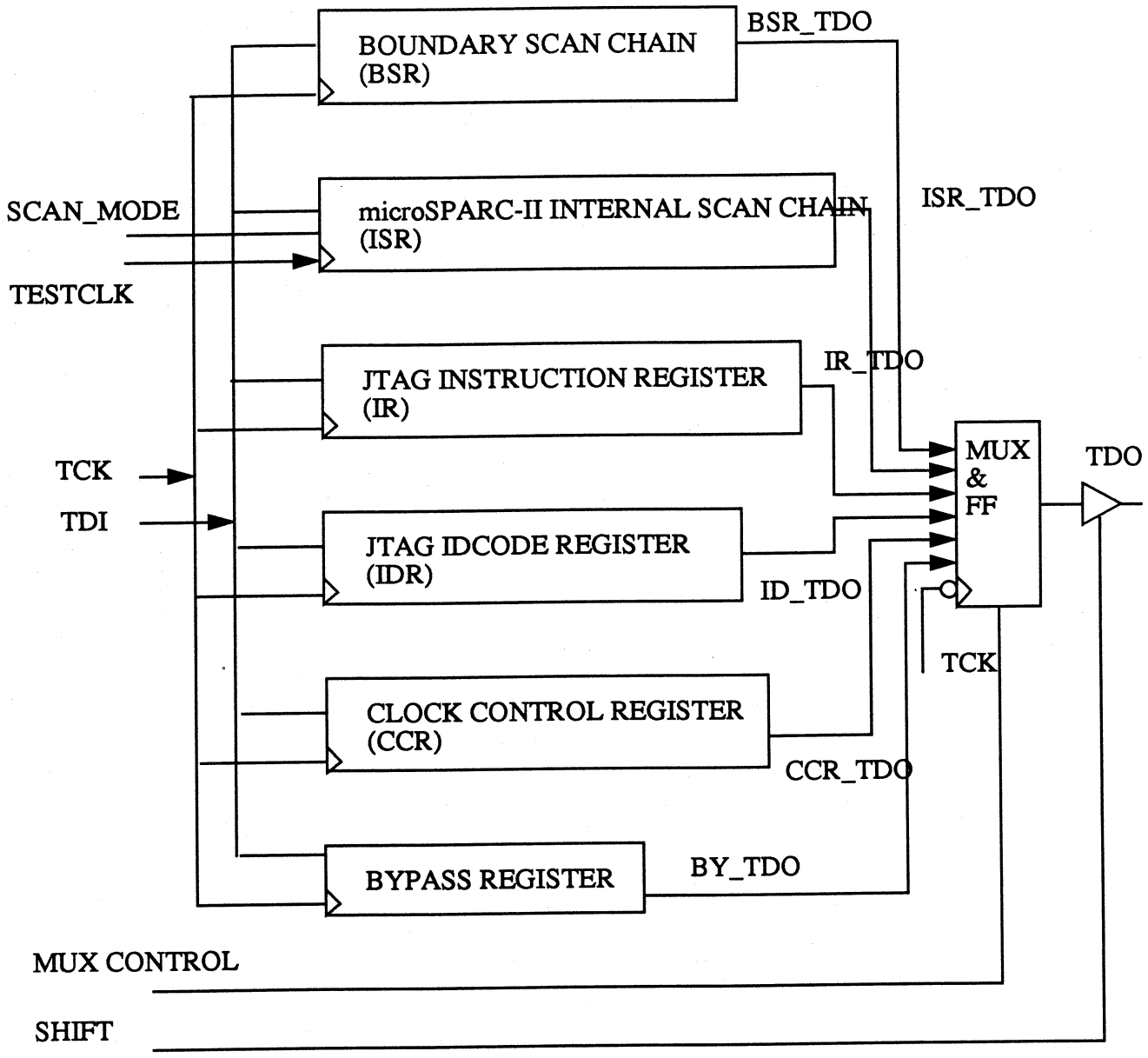


Figure 10.5 - JTAG DATA & INSTRUCTION REGISTERS



10.14 CLK_RST TAP Instruction

The microSPARC-II clk_cntl block is a collection of non-scanned logic which generates the various clock waveforms which are used both on and off the microSPARC-II chip. Although this logic is not directly scannable, microSPARC-II implements a private TAP instruction for initializing the state of the flipflops in the rl_clk_cntl block. This instruction is intended for use by a tester, since it requires precise control of the waveforms driven onto the ext_clk1/ext_clk2 microSPARC-II input pins.

The instruction mnemonic is CLK_RST, and its binary opcode is 110000. Its behavior is identical to that of the BYPASS instruction, except that the internal signal clk_rst_l is asserted whenever the CLK_RST opcode appears on the TAP instruction register output latch (i.e. starting at the falling edge of jtag_ck when the TAP state machine is in the Update-IR state - see IEEE Std 1149.1 for details of the TAP state machine operation). While clk_rst_l is asserted, all of the flipflops in rl_clk_cntl will be synchronously reset at the rising edge of the high-speed input_clock.

It is intended that the CLK_RST operation be used only when the microSPARC-II pll_byp_l input pin is driven to 0, i.e. when the internal phase-locked-loop is being bypassed. In that mode, input_clock is equal to the XOR of the ext_clk1 and ext_clk2 input pins. Here is an algorithm which can be used to reset rl_clk_cntl to a known state:

0) Apply clocks to jtag_ck, drive jtag_tdi=1, and drive pll_byp_l=0 for the duration of the test. Drive ext_clk1=0 and ext_clk2=0 through Step 3 below, with the exception of a single 0->1->0 pulse on ext_clk1 in step 3.

1) Assert jtag_trst_l, then de-assert it, to reset the TAP controller.

2) Apply this sequence of values to jtag_ms, applying a new value at each negative edge of jtag_ck (the number below each value is a cycle count, for reference):

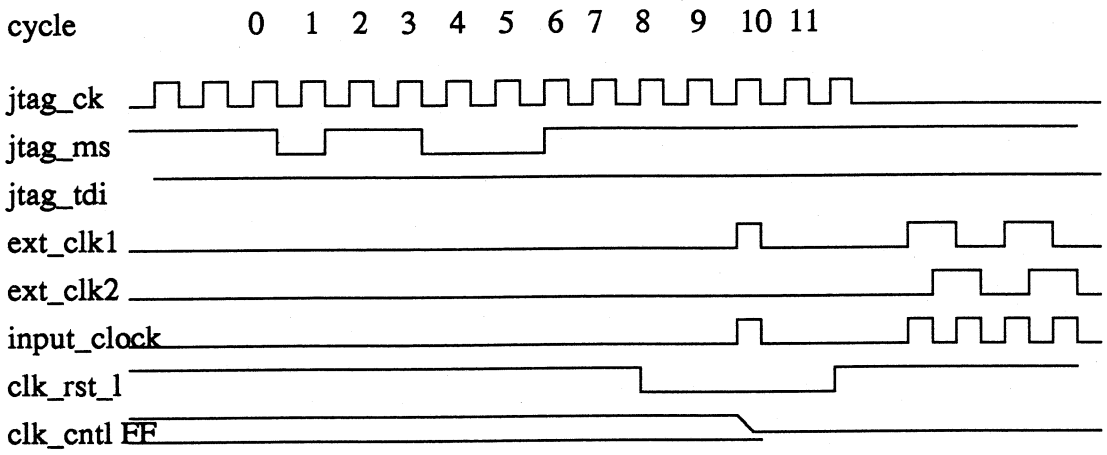
(1, . . . 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, . . .)
0 1 2 3 4 5 6 7 8 9 10 11

Note that in cycle 5, the IR is parallel-loaded with 000001. (see rule 6.1.1.d). In cycle 6 and 7, ones are shifted into the MSB end of the IR. The result is a 110000 in the IR>

3) In cycle 10 of the sequence above, apply a single 0->1->0 pulse o ext_clk1. The rising edge of this pulse will reset the rl_clk_cntl block.

4) After cycle 11 of the sequence above, rl_clk_cntl has been reset and the TAP controller is in the Test-Logic-Reset state. You may now assert jtag_trst and being applying clocks to ext_clk1 and ext_clk2 to start the test.

Figure 10.6 - Jtag Clk Reset operation



Chapter 11 Error Handling

The microSPARC-II CPU must detect and handle many kinds of errors and exceptions. The SPARC IU is interrupted by some type of trap in all CPU error cases. DMA masters other than the CPU should cause their own IU trap via the SBus interrupt mechanism. Physical address references to nonexistent addresses in any address space will either return garbage or cause timeouts. The following preliminary list attempts to describe what happens under various circumstances.

Table 50 - Error Summary

Error	Initiator	Result Summary
Memory Parity Error	Instruction Memory Access	set PE, FT=5, L, AT in SFSR cause Instruction Access Error trap (D stage + 1)
	IU, FPU Read Memory Access	set PE, ERR, CP, TYPE in MFSR save PA in MFAR cause L15 interrupt
	IU, FPU Write Byte, Half-word Memory Access (Read-modify-write)	set PE, ERR, CP, TYPE in MFSR save PA in MFAR cause L15 interrupt
(Translation Error)	Tablewalk on Instruction Memory Access	set PE, FT=4, L, AT in SFSR cause Instruction Access Error trap (D stage)
(Translation Error)	Tablewalk on IU, FPU Data Memory Access	set PE, FT=4, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
	IO DMA Read Memory Access	return SBus Error Acknowledge set PE, ERR in MFSR save PA in MFAR, cause L15 interrupt
	IO DMA Write Byte, Half-word Memory Access (Read-modify-write)	return SBus Error Acknowledge set PE, ERR in MFSR save PA in MFAR, cause L15 interrupt
	Tablewalk on IO DMA Memory Access	return SBus Error Acknowledge set PE, ERR in MFSR save PA in MFAR, cause L15 interrupt
SBus Controller Time Out	CPU SBus Read Access	set TO, FT=5, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
	CPU SBus Write Access	set TO, ERR, SIZE, ~RD, FAV in AFSR save PA in AFAR cause L15 interrupt
	IO DMA Access	return SBus Error Acknowledge
SBus Late Error (Ack)	CPU SBus Read Access	set LE, ERR, SIZE, RD, FAV in AFSR
	CPU SBus Write Access	set LE, ERR, SIZE, FAV(sometimes) in AFSR

Error	Initiator	Result Summary
SBus Error Acknowledge	CPU Read Access	set BE, FT=5, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
	CPU Write Access	set BE, ERR, SIZE, ~RD, FAV in AFSR, save PA in AFAR cause L15 interrupt using CP_STAT
Invalid Address Error	IO DMA PTE Access (IO PTE V bit = 0)	return SBus Error Acknowledge
	ET=0 during Tablewalk on Instruction Memory Access	set FT=1, L, AT in SFSR cause Instruction Access Exception trap (D stage)
	ET=0 during Tablewalk on IU, FPU Data Memory Access	set FT=1, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Translation Error	ET=3 during Tablewalk on Instruction Memory Access	set FT=4, L, AT in SFSR cause Instruction Access Error trap (D stage)
	ET=3 during Tablewalk on IU, FPU Data Memory Access	set FT=4, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
Control Space Error	CPU Invalid ASI Access	set FT=5, L, FAV, CS in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
	CPU Invalid Size of Access	set FT=5, L, FAV, CS in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
	CPU Invalid Virtual Address during ASI requiring VA	set FT=5, L, FAV, CS in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Privilege Violation Error (S bit and not ACC 6,7)	IU Instruction Memory Access	set FT=3, L, AT in SFSR cause Instruction Access Exception trap (D stage)
Privilege Violation Error (ACC and ASI checked)	IU, FPU Data Memory Access	set FT=3, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Protection Error (Memory page ACC and the ASI are checked)	IU, FPU Data Memory Access	set FT=2, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Protection Error (Memory page ACC is checked)	IU, FPU Data Memory Access	set FT=2, L, AT, FAV in SFSR cause Instruction Access Exception trap (D stage)
Protection Error (Write to read only page)	IO DMA Write	return SBus Error Acknowledge

Chapter 12 ASI Map

This chapter describes the microSPARC-II ASI map.The Address Space Identifier (ASI) is appended to the virtual address by the SPARC IU when it accesses memory. The ASI encodes whether the processor is in supervisor or user mode, whether an access is to instruction or data memory, and is used to perform other internal cpu functions.

12.1 Overview

The table below lists all of the ASI values supported in a microSPARC-II system. Only the least significant 6 bits of the ASI are decoded.

Table 51 - ASI's Supported by microSPARC-II

ASI	Function	Acc	Size	ASI	Function	Acc	Size
00	Reserved	-	-	10	Flush I&D Cache Line (page)	W	Single
01-02	Unassigned	-	-	11	Flush I&D Cache Line (seg)	W	Single
03	Ref MMU Flush/Probe	R/W	Single	12	Flush I&D Cache Line (reg)	W	Single
04	MMU Registers	R/W	Single	13	Flush I&D Cache Line (ctxt)	W	Single
05	Unassigned	-	-	14	Flush I&D Cache Line (user)	W	Single
06	Ref MMU Diagnostics	R/W	Single	15-16	Reserved	-	-
07	Unassigned	-	-	17-1C	Unassigned	-	-
08	User Instruction	R/W	All	1D-!E	Reserved	-	-
09	Supervisor Instruction	R/W	All	1F	Unassigned	-	-
0A	User Data	R/W	All	20	Ref MMU Bypass	R/W	All
0B	Supervisor Data	R/W	All	21-2F	Reserved	-	-
0C	Instruction Cache Tag	R/W	Single	30-38	Unassigned	-	-
0D	Instruction Cache Data	R/W	Single	39	Data Cache Diag Register	R/W	Single
0E	Data Cache Tag	R/W	Single	3A-3F	Unassigned	-	-
0F	Data Cache Data	R/W	Single	40-FF	Reserved	-	-

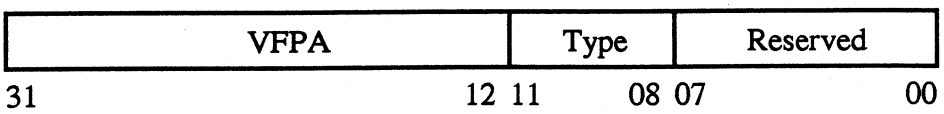
ASI Descriptions:

ASI=0x00 **Reserved** - This space is architecturally reserved.

ASI=0x01-0x02 **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x03 **Ref MMU Flush/Probe** - This space is used for a flush or probe operation. The Virtual Address is decodes as follows.

Figure 12.0 - TLB Flush or Probe Address Format



Field Definitions:

Virtual Flush or Probe Address (VFPA) - This field is the address that is used to index into TLB. Depending on the type of flush or probe not all 20 bits are significant.

Type - This field specifies the extent of the flush or the level of the entry probed.

Reserved - These bits are ignored. They should be set to zero.

A flush is caused by a single STA instruction and a probe by a single LDA instruction.

Flushes are used to maintain TLB consistency by conditionally removing one or more page descriptors. These conditions vary as shown.

Table 52 - TLB Entry Flushing

Type	Flush	PTE Match Criteria
0	Page	((ACC $\geq$ 6) OR CID match) AND VA[31:12] match
1	Segment	((ACC $\geq$ 6) OR CID match) AND VA[31:18] match
2	Region	((ACC $\geq$ 6) OR CID match) AND VA[31:24] match
3	Context	(ACC $\leq$ 5) AND CID match
4	Entire	None (Entire TLB Flush)
5 to F	Reserved	-

Probes cause the MMU to perform a table walk. The table walk will stop when a PTE has been reached as shown.

Table 53 - CPU TLB Entry Probing

TYPE	If No Memory Errors Occur																Memory Error
	Level-0 Entry Type				Level-1 Entry Type				Level-2 Entry Type				Level-3 Entry Type				
	pte	res	inv	ptp	pte	res	inv	ptp	pte	res	inv	ptp	pte	res	inv	ptp	
0(page)	0	0	0	=>	0	0	0	=>	0	0	0	=>	X	0	X	0	0
1(segment)	0	0	0	=>	0	0	0	=>	X	0	0	X	--	--	--	--	0
2(region)	0	0	0	=>	X	0	X	X	--	--	--	--	--	--	--	--	0
3(context)	X	0	X	X	--	--	--	--	--	--	--	--	--	--	--	--	0
4(entire)	X	0	0	=>	X	0	0	=>	X	0	0	=>	X	0	0	0	0
5-0xF	(undefined)																

ASI=0x04

MMU Registers - This space is used to read and write internal MMU registers using the Virtual Address to reference them. Single word accesses only should be used, others result in an error.

Table 54 - Address Map for MMU Registers

VA[12:08]	Register
00	Control Register
01	Context Table Pointer Register
02	Context Register
03	Synchronous Fault Status Register
04	Synchronous Fault Address Register
05-0F	Reserved
10	TLB Replacement Control Register
11-12	Reserved
13	Synchronous Fault Status Register**
14	Synchronous Fault Address Register**
15-1F	Reserved
	**Writeable for diagnostic purposes

VA bits [31:13] are zero. VA bits [07:00] are ignored and should be set to zero by software.

ASI=0x05 **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x06 **Ref MMU Diagnostics** - Diagnostic reads and writes can be made to the 64 TLB entries using the virtual address to specify which entry and whether the PTE or Tag section is to be referenced.

ASI=0x07 **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x08 **User Instruction** - This space is defined and reserved by SPARC for user instructions.

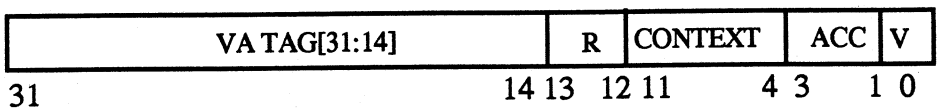
ASI=0x09 **Supervisor Instruction** - This space is defined and reserved by SPARC for supervisor instructions.

ASI=0x0A **User Data** - This space is defined and reserved by SPARC for user data.

ASI=0x0B **Supervisor Data** - This space is defined and reserved by SPARC for supervisor data.

ASI=0x0C **Instruction Cache Tag** - This space is used for reading and writing instruction cache tags by using the LDA and STA instructions at virtual addresses in the range of 0x0 to 0x03FFF on modulo-32 boundaries.

Figure 12.1 - Instruction Cache Tag Entry



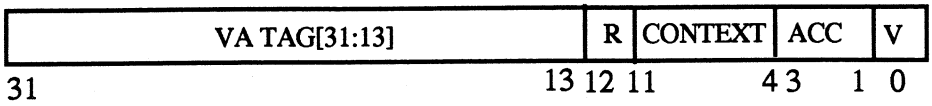
ASI=0x0D

Instruction Cache Data - This space is used for reading and writing instruction cache data by using the LDA and STA instructions at virtual addresses in the range of 0x0 to 0x03FFF.

ASI=0x0E

Data Cache Tag - This space is used for reading and writing data cache tags by using the LDA and STA instructions at virtual addresses in the range of 0x0 to 0x01FFF on modulo-16 boundaries.

Figure 12.2 - Data Cache Tag Entry



ASI=0x0F

Data Cache Data - This space is used for reading and writing data cache data by using the LDA and STA instructions in ASI 0xF at virtual addresses in the range of 0x0 to 0x01FFF.

ASI=0x10-0x14

Flush I & D Cache Line - These spaces are used to flush single cache lines by using the STA instruction to one of these spaces.This results in a single line being removed from both I and D caches.

A cache line is flushed if it meets the minimum criteria given in the following table. "S" is the supervisor bit, "U" is the inverse of S, "CNTXT" is the matching of the context register and Tag context, and VA[31:xx] is a comparison based on the virtual address tag.

Table 55 - Flush Criteria for ASI 0x10-0x14

ASI[2:0]	Flush Type	Compare Criterion
0	Page	(S or CNTXT) and VA[31:12]
1	Segment	(S or CNTXT) and VA[13:18]
2	Region	(S or CNTXT) and VA[31:24]
3	Context	U and CNTXT
4	User	U

Table 55 - Flush Criteria for ASI 0x10-0x14

ASI[2:0]	Flush Type	Compare Criterion
5,6	reserved	-

ASI=0x15-0x16 **Reserved** - This space is architecturally reserved.

ASI=0x17-0x1C **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x1D-0x1E **Reserved** - This space is architecturally reserved.

ASI=0x1F **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x20 **Ref MMU Bypass** - This space can be used to access an arbitrary physical address. It is particularly useful before the MMU or main memory have been initialized. The MMU does not perform an address translation rather a physical address is formed from the least significant 31 bits of the Virtual Address (PA[30:00] := VA[30:00]). Accesses in bypass mode are not cacheable.

ASI=0x21-0x2F **Reserved** - This space is architecturally reserved.

ASI=0x30-0x38 **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x39 **Data Cache Diag Register** - This space is used to read and write internal Data Cache registers using the Virtual Address to reference them. Single word accesses only should be used, others result in an error.

ASI=0x3A-0x3F **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x40-0xFF	**Reserved** - Since the 2 high order bits are not decoded these encodings should not be used. If they are used the two upper bits are ignored and only the lower 6 bits will be decoded.

Chapter 13 References

1. The SPARC Architecture Manual, Version 8, of December 11, 1990
2. IEEE P1496 Proposed Standard for Sbus 3-24-92 draft 0.5
3. The SuperSPARC Microprocessor User's Manual, Rev. 1.00 April 1994
4. Sun-4M System Architecture Proposal, Rev. 1.1 of March 23, 1990
5. IEEE standard 1149.1, IEEE Standard Test Access Port and Boundary Scan Architecture, IEEE, 1990
6. I/O Cell Circuit for Wire and Tab with Jtag Function (Preliminary) Rev 0.02 Fujitsu March 26, 1992

Appendix A microSPARC-II Local Graphics Bus

A.1 Introduction

The Local Graphics Bus is a memory-level interconnect protocol between the processor (microSPARC-II), system memory, and graphics subsystems. The Local Graphics Bus provides high bandwidth, low latency, and slave-only access to graphics by placing the graphics interface directly on the system memory bus in a uni-processor environment.

The principal features of the Local Graphics Bus are:

- A 64-bit data path
- A 28-bit physical address per slave
- Clock rate of up to 75 MHz
- Synchronous operation, except for interrupts
- One, two, four, or eight-byte transfers
- Single interrupt line per slave

The Local Graphics Bus provides high-speed access between the processor, system memory, and graphics devices. All accesses to the system memory and Local Graphics Bus slaves are controlled by microSPARC-II processor. The Local Graphics Bus Controller (or simply Controller) in this document refers to the Local Graphics Bus Controller which is integrated within the microSPARC-II chip. Any Local Graphics Bus Slave can request an interrupt of the Local Graphics Bus Controller, but only the Controller can perform read or write accesses. Figure A.1 shows a block diagram of the Local Graphics Bus.

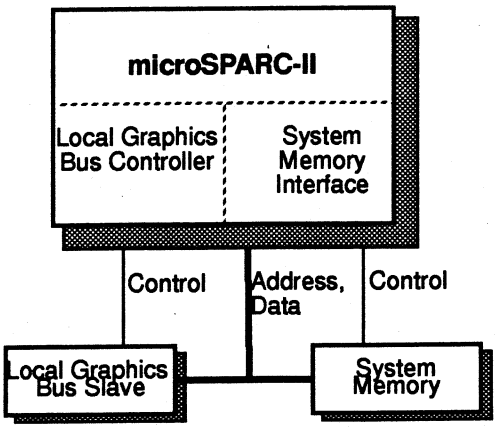


Figure A.1 - Local Graphics Bus Block Diagram

System Memory Interface

Since the Local Graphics Bus shares the system memory address and data lines, the Local Graphics Bus Controller and system memory interface must arbitrate for use of these common lines. The memory interface is integrated within the microSPARC-II chip.

Local Graphics Bus Controller

The Local Graphics Bus Controller is the single master of the Local Graphics Bus and is responsible for initiating each Local Graphics Bus cycle. The Local Graphics Bus Controller is integrated within the microSPARC-II chip.

Local Graphics Bus Slave

The Local Graphics Bus Slave responds to requests given by the Local Graphics Bus Controller. The Local Graphics Bus Slave may interrupt the Local Graphics Bus Controller via a single asynchronous interrupt line.

The Local Graphics Bus may have multiple Slave devices. Each Local Graphics Bus Slave neither knows nor cares about other Local Graphics Bus Slaves.

Local Graphics Bus Interface

Table A.1 shows the interface signals that connects the microSPARC-II chip to the a Local Graphics Bus Slave. These signals are either generated by the Controller (microSPARC-II) or the Local Graphics Bus Slave. DB[63:0]) bus may be driven by either microSPARC-II or the slave depending on the type of bus cycle.

Table A.1 - Local Graphics Bus Interface Signals

Signal Name	Description	Driven By
CLK	Clock	Controller
AEN	Address enable	Controller
LO_ADDR	Low address select	Controller
WRITE_L	Read/write select	Controller
AB[14:0]	Address/Byte Mask bus	Controller
P_REPLY[1:0]	Port reply	Slave
S_REPLY[1:0]	System reply	Controller
DB[63:0]	Data bus	Controller/Slave
RESET_L	Reset	Controller
INT_L	Interrupt	Slave

A.2 Basic Local Graphics Bus Cycle

A Local Graphics Bus Slave has a minimum four-deep request FIFO that holds address, size, and (for write requests) data. There may be at most one outstanding read request in the FIFO at any time. Write requests sent to the Slave should be saved in the FIFO and later acknowledged. Read requests must allow the FIFO to drain before responding.

The Local Graphics Bus Controller must assure a FIFO depth of four slots, but may optionally probe the Slave after a power-on reset to determine the exact size of its FIFO. The Controller throttles access requests to the Slave. There are no FIFO full or FIFO empty signals between the Controller and Slave.

A complete Local Graphics Bus cycle consists of two major phases: address and data. These two phases may operate independently of each other. Addresses of successive accesses may be sent to the Slave without regard to the associated data, as long as the Slave's FIFOs are not overloaded. Data transfers are completed using the Reply control signals, allowing FIFO entries in the Slave to again be available.

Address Cycles

Local Graphics Bus address cycles send physical address, access size, and access direction in two separate cycles. These cycles are controlled by the AEN (address enable) signal. The cycle type is denoted by the state of the LO_ADDR signal.

In address cycle 0, the upper bits of the physical address are placed on the AB (address bus) lines. In address cycle 1, the lower bits of the physical address and the access size are placed on the AB lines, while the access direction (read or write) is defined by the WRITE_L signal.

For subsequent accesses, address cycle 0 need not be repeated if the upper physical address bits do not change. The Slave must latch this data and use it until the Controller sends the next address cycle 0. This functionality is similar to the "Page Mode" feature of DRAMs.

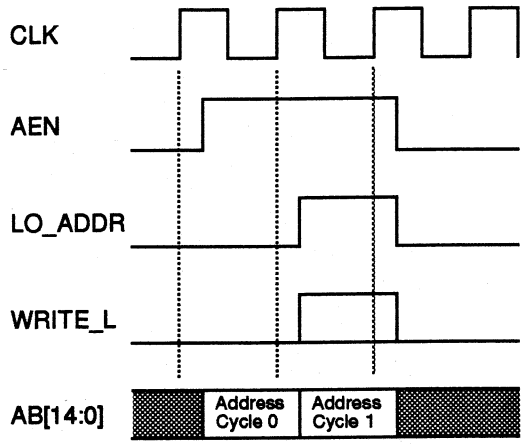


Figure A.2 - Address Cycles

Data Cycles

Each time an address cycle 1 is sent, the read or write command to the Slave should be considered launched and a data cycle may be initiated.

The Local Graphics Bus cycle must complete within 2048 Local Graphics Bus clock cycles of the launching address cycle. The Local Graphics Bus Controller will time-out if the Slave has not responded with the appropriate P_REPLY (port reply) code.

Slaves must always acknowledge all accesses to its entire address range using the appropriate P_REPLY code (read single or write single). Since the Local Graphics Bus does not specify an error acknowledge, Slaves may provide a readable location which identifies when error accesses have occurred. These error types may include out-of-bounds access, unsupported size-type access, and so on. Slaves may also interrupt the Controller to identify error accesses.

Write

The Controller initiates the write data cycle by placing the "Write Single" code (10) on the S_REPLY (system reply) lines. This may happen in the same clock as the launching address cycle 1, or any clock cycle following. Once sent, the Controller will place the write data on the DB (data bus) lines in the next clock cycle. The Slave must acknowledge the write by placing the "Write Single" code (10) on the P_REPLY lines in the next clock cycle, or later.

Read

In the **read** cycle, the CPU issues a "Read Single" code (11) on the S_REPLY lines in the same clock cycle as the launching address cycle 1. When the Slave is ready to acknowledge the read, the Slave drives the data on the DB lines and a "Read Single" code (11) on the P_REPLY lines in the same clock cycle. This mode has the advantage of doing a best-case read in two clock cycles.

Local Graphics Bus Timeout

The Local Graphics Bus Controller will support a timeout mechanism to prevent the system from waiting on a broken or absent Local Graphics Bus Slave device. The timeout period for an access is measured from the positive edge of the CLK which launches an Local Graphics access (address cycle 1). This period is defined as 2048 CLK (clock) cycles (this must be greater than 10 microseconds) before the corresponding P_REPLY is returned by the slave.

If a timeout occurs, the Controller will abort the current cycle and all outstanding Local Graphics Bus cycles.

Local Graphics Bus Latency

For **read** cycles, an Local Graphics Bus Slave should respond to accesses with an average latency of 1.0 microseconds, and a worst case latency of 2.0 microseconds.

To enforce this, the Controller must hold off write requests until a slot is available in the Slave's write FIFO. The Controller must also hold off read requests until the Slave's write FIFO has zero or one writes pending.

The threshold will be settable at the user level and live as part of the physical address control space on its own page. If the threshold is zero, the latency for a read is set by the Slave's maximum read time. If the threshold is one, the maximum latency is for one read and one write retirement. Default at power on is zero.

If an Local Graphics Bus Slave implements features which would cause a write-read combination to violate the above worst case latency, it should support a software polling mechanism. Immediate access register(s) should be provided to respond to inquiries about the status of the Slave device.

Local Graphics Bus interrupt latency should nominally be less than 10 microseconds and less than 50 microseconds worst case.

A.3 Local Graphics Memory Map

Each slave occupies 256M bytes of address space in a system memory map. The Boot PROM always begins at location zero.

A.4 Local Graphics Bus Interconnect

Figure A.3 shows the Local Graphics Bus interconnect and all of the associated signals. In the case where a system has multiple Local Graphics Bus slaves, some signals are required to be unique to each one. These are noted as *radial* (R). For an efficient implementation, other signals may be common between connectors. These are noted as *bused* (B).

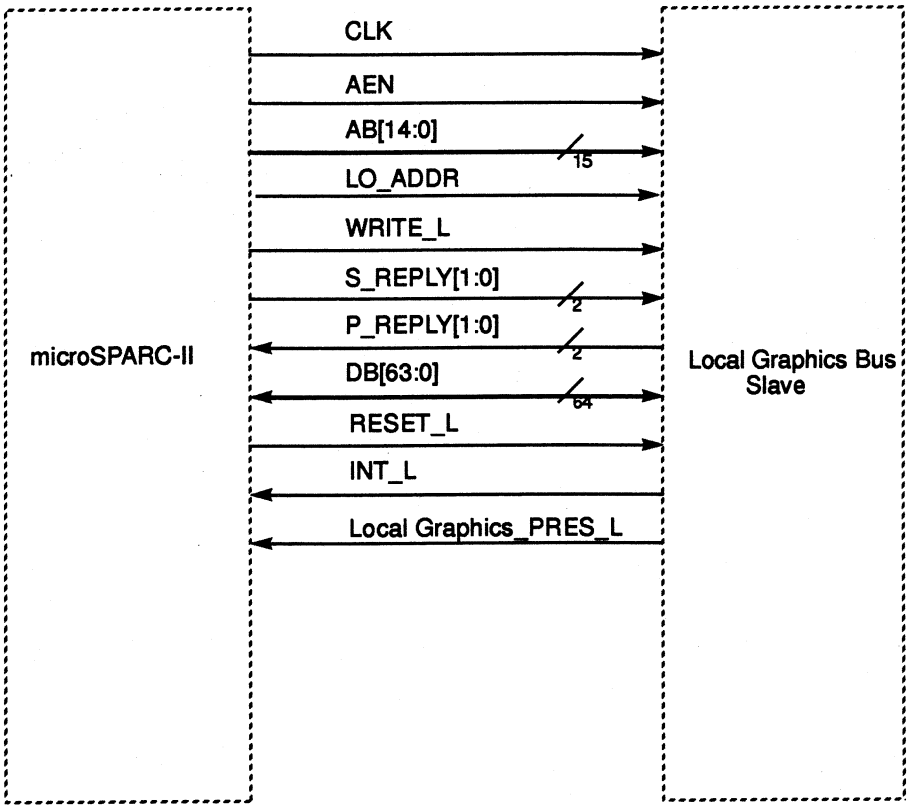


Figure A.3 - Local Graphics Bus *Signals*

Table A.2 - Local Graphics Bus Signal Summary

Signal Name	I/O	Description	Driven By	Multiple Slaves[1]
Bus Signals:				
CLK	I	Clock	Controller	B
AEN	I	Address enable	Controller	R
LO_ADDR	I	Low address select	Controller	B
WRITE_L	I	Read/write select	Controller	B
AB[14:0]	I	Address/Byte Mask bus	Controller	B
P_REPLY[1:0]	O	Port reply	Slave	R
S_REPLY[1:0]	I	System reply	Controller	R
DB[63:0]	I/O	Data bus	Controller/ Slave	B
RESET_L	I	Reset	Controller	B
INT_L	O	Interrupt	Slave	R
Local Graphics_ PRES_L	O	Device present	Slave	R

1. R = Radial, B = Bused.

A.5 Local Graphics Bus Signals

This section provides detailed descriptions of the Local Graphics Bus signals.

CLK

Local Graphics Bus clock. This signal is generated by microSPARc-II chip and its frequency is within the range of 25 MHz to 42 MHz. The clocks may be used in differential form to improve the common mode noise immunity at high clock rates. The maximum clock frequency in the single-ended mode is 42 MHz. Typical system clock rates will run at 20 MHz and higher.

AEN

Address enable. When asserted (high), this signal indicates that there is a command request and there is valid data on the AB bus and the LO_ADDR and WRITE_L lines. The AEN signal also qualifies read cycles.

LO_ADDR

Low address. This signal is qualified by the AEN (address enable) signal. LO_ADDR defines which address cycle the Controller is sending on the address bus and write lines. When driven to 0, this signal indicates address cycle 0 (high address bits). When driven to 1, this signal indicates address cycle 1 (low address bits and byte mask).

WRITE_L

Write. This signal is qualified by AEN (address enable) and is only valid during address cycle 1. When driven to 0, this signal indicates a write request. When driven to 1, this signal indicates a read request.

AB[14:0]

Address/Byte Mask bus. The address/byte mask bus contains the multiplexed physical address of where the data should be written to or read from and the byte mask. This data is multiplexed over two address cycles, as shown in Table A.3

Table A.3 - Address Bus Multiplexing

Address Bus	Cycle 0	Cycle 1
LO_ADDR	0	1
WRITE_L	X	R/W
AB[14]	PA[27]	BM[3]
AB[13]	PA[26]	BM[2]
AB[12]	PA[25]	BM[1]
AB[11]	PA[24]	BM[0]
AB[10]	PA[23]	PA[13]
AB[9]	PA[22]	PA[12]
AB[8]	PA[21]	PA[11]
AB[7]	PA[20]	PA[10]
AB[6]	PA[19]	PA[9]
AB[5]	PA[18]	PA[8]
AB[4]	PA[17]	PA[7]
AB[3]	PA[16]	PA[6]
AB[2]	PA[15]	PA[5]
AB[1]	PA[14]	PA[4]
AB[0]	X	PA[3]

PA = Physical address
BM[3:0] = Byte mask bits
X = don't care
R/W = Read or write

Byte Mask (BM) Bits

The byte mask bits, BM[3:0], indicate the transfer size, as shown in Table A.4,

Table A.4 - Byte Mask (BM) Bits

	Data								
BM[3:0]	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]	Comment
0000	R/W	--	--	--	--	--	--	--	Byte access 0
0001	--	R/W	--	--	--	--	--	--	Byte access 1
0010	--	--	R/W	--	--	--	--	--	Byte access 2
0011	--	--	--	R/W	--	--	--	--	Byte access 3
0100	--	--	--	--	R/W	--	--	--	Byte access 4
0101	--	--	--	--	--	R/W	--	--	Byte access 5
0110	--	--	--	--	--	--	R/W	--	Byte access 6
0111	--	--	--	--	--	--	--	R/W	Byte access 7
1000	R/W	R/W	--	--	--	--	--	--	Half word 0
1010	--	--	R/W	R/W	--	--	--	--	Half word 1
1100	--	--	--	--	R/W	R/W	--	--	Half word 2
1110	--	--	--	--	--	--	R/W	R/W	Half word 3
1001	R/W	R/W	R/W	R/W	--	--	--	--	Word 0
1101	--	--	--	--	R/W	R/W	R/W	R/W	Word 1
1011	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Double word
1111	--	--	--	--	--	--	--	--	Reserved

Note: The BM codes in the left column are not in absolute numerical order.

Multiplexed Addresses

After any reset issued to the Slave, the Controller will establish a full physical address for the first access request to the Slave. Since the physical address is multiplexed across two address cycles, both address cycle 0 and address cycle 1 must be issued to the Slave.

The Slave must always latch the data from address cycle 0 (upper physical address bits) to combine them with the lower physical address bits in each following address cycle 1. Another address cycle 0 would only need to be issued when the data access crosses the "page boundary" and the upper address bits need to be modified.

The Local Graphics Bus Controller is allowed to issue extraneous address cycle 0's, but it is recommended that this be minimized for better system performance.

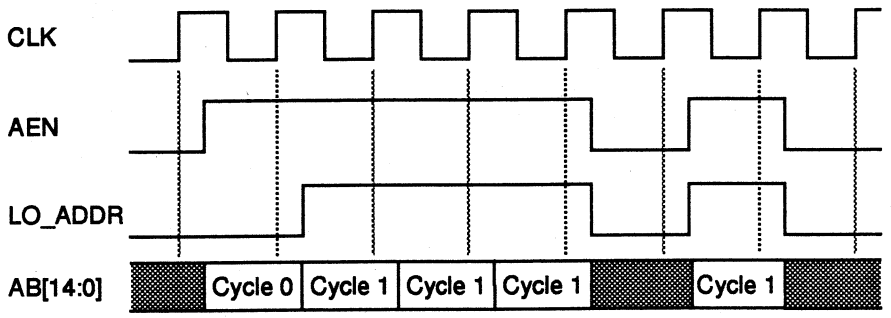


Figure A.4 - Multiplexed Addresses

P_REPLY[1:0]

Port reply. These output signals indicate that the data has been processed (removed) from the write buffer on writes or that the read data is available in the read latch.

Table A.5 - P_REPLY[1:0] Signals

P_REPLY[1]	P_REPLY[0]	Type of Access
0	0	Idle
0	1	Reserved
1	0	Write single (CE) P_WAS
1	1	Read single (OE) P_RAS

S_REPLY[1:0]

System reply. These signals indicate that the Local Graphics Bus has been selected and the type of access.

Table A.6 - S_REPLY[1:0] Signals

S_REPLY[1]	S_REPLY[0]	Type of Access
0	0	Idle
0	1	Idle
1	0	Write single (CE) S_WRS
1	1	Read single (OE) S_SRS

On writes, the data follows in the following cycle. In read cycles, the data is enabled onto the bus in the same cycle as the P_REPLY signal is driven. See Figure A.9.

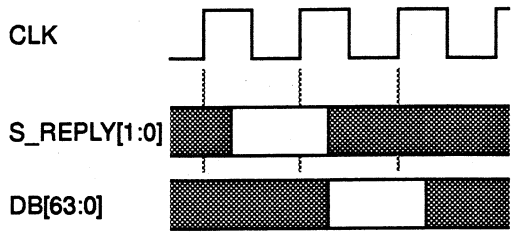


Figure A.5 - S_REPLY[1:0] Signal

DB[63:0]

Data bus. The data being transferred between the microSPARc-II and the Local Graphics Bus Slave. Every Local Graphics Bus must have all 64 data bus signals, over which data is transferred. The Local Graphics Bus supports four primary data formats:

- Bytes, which consist of eight data bits
- Half words, which consist of 16 data bits
- Words, which consist of 32 data bits
- Double words, which consist of 64 data bits

By convention, the least-significant bit of the data bus is DB[0] while the most-significant bit is DB[63]. These pins have internal pullups.

The Local Graphics Bus uses what is commonly called *big-endian* addressing. As shown in Figure A.6, big-endian addressing means that the significance of bytes in a half-word, word, or double-word decreases as the address increases. The byte ordering is specified by the Byte Mask bits in the address.

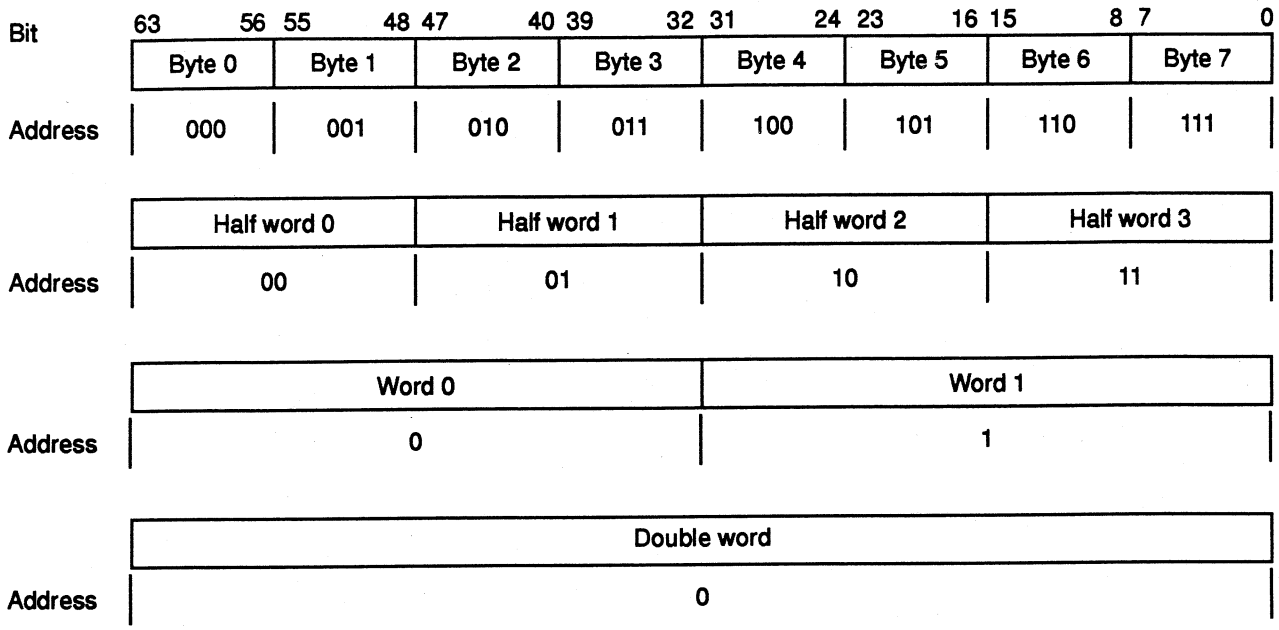


Figure A.6 - Data Bus Byte Ordering

RESET_L

Local Graphics Bus reset. The $\overline{\text{RESET\_L}}$ signal properly initializes all Local Graphics Slaves after power-up and system reset. In all cases, RESET_L must be asserted for at least 512 clock cycles before being unasserted. In the case of power-on, RESET_L must be stable before these 512 clock cycles begin. The leading edge of RESET_L may or may not meet setup times with respect to CLK. The trailing edge of RESET_L must meet setup and hold times with respect to CLK. The Local Graphics Bus controller may keep RESET_L asserted for more than 512 clock cycles, if desired.

Upon detecting the assertion of RESET_L, a Local Graphics Bus Slave must perform whatever internal operations are required to initialize itself. While RESET_L is asserted, a Local Graphics Bus Slave must not drive bused signals and must drive radial signals to an inactive state.

INT_L

Local Graphics Bus interrupt. The Slave uses this signal to signal the Controller that some event has occurred and requires servicing. This signal is unique to each Slave and need not be bused. This signal should be driven with an open-drain driver.

Before asserting the interrupt, the Slave must set a bit in an internal register to indicate the cause of the interrupt. The interrupt is serviced by microSPARC-II after reading this bit. The bit may be cleared by reading it or by an explicit write by microSPARC-II. Once cleared, the Slave must then unassert the interrupt signal.

This feature must be maskable and its default mode is disabled.

Local GraphicsPRES_L

Local Graphics device present. This pin is used to signify that an Local Graphics Slave device is present in this physical slot. This signal is static and is only checked by the system at probe time. The Controller should have a pullup on this signal. Slave cards should then simply ground this pin.

A.6 Local Graphics Bus Timing Diagrams

This section shows the timing diagrams for Write and Read cycles on the Local Graphics Bus for microSPARC-II.

Write Cycle

In a write, the Controller places the address on AB[14:0] and asserts AEN. This example shows a two-cycle address, so the Controller asserts LO_ADDR for address cycle 1 (the low address). The Controller asserts S_REPLY either simultaneous with the second address or up to n cycles later.

The Controller places the write data on DB[63:0] on the next cycle after receiving the S_REPLY write single (10) code. The Slave acknowledges receipt of the data by asserting the P_REPLY write single (10) code.

Figure A.7 shows the timing for a fast write. Figure A.8 shows the timing for a slow write.

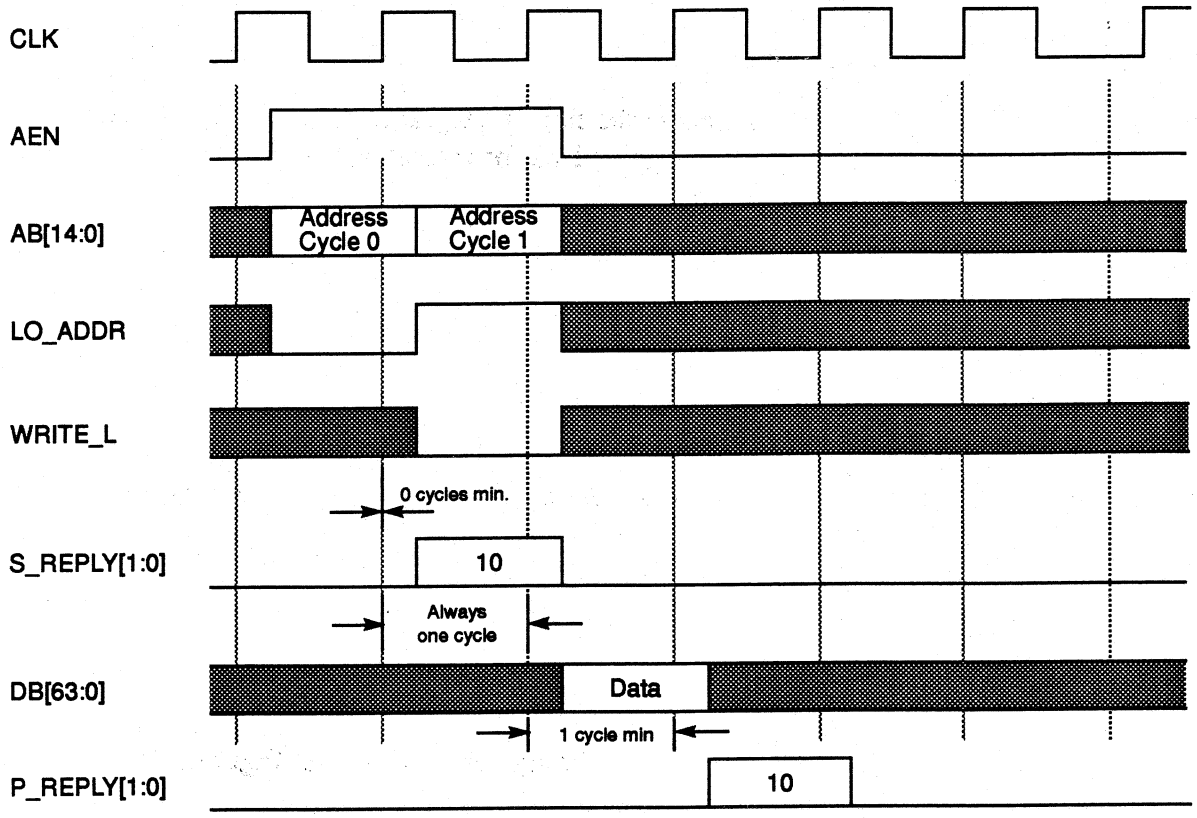


Figure A.7 - Fast Write Timing

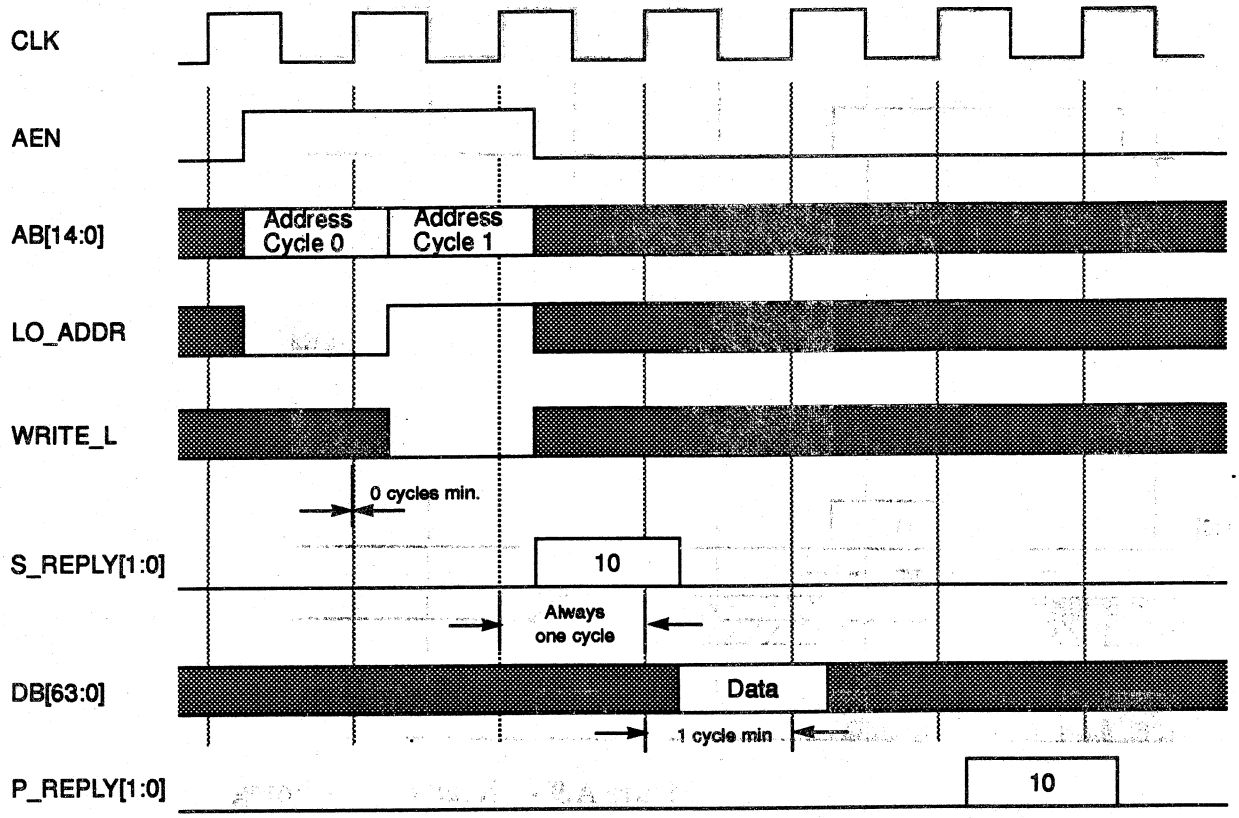


Figure A.8 - Slow Write Timing

Read Cycle

Figure A.9 shows the read timing. In the read, the Controller places the address on AB[14:0] and asserts AEN. On the second address cycle (address cycle 1), the Controller deasserts WRITE_L and asserts LO_ADDR and the S_REPLY read single (11) code.

As soon as it has the data available, the Slave responds to the read request by placing the requested read data on DB[63:0] and asserting the P_REPLY read single (11) code.

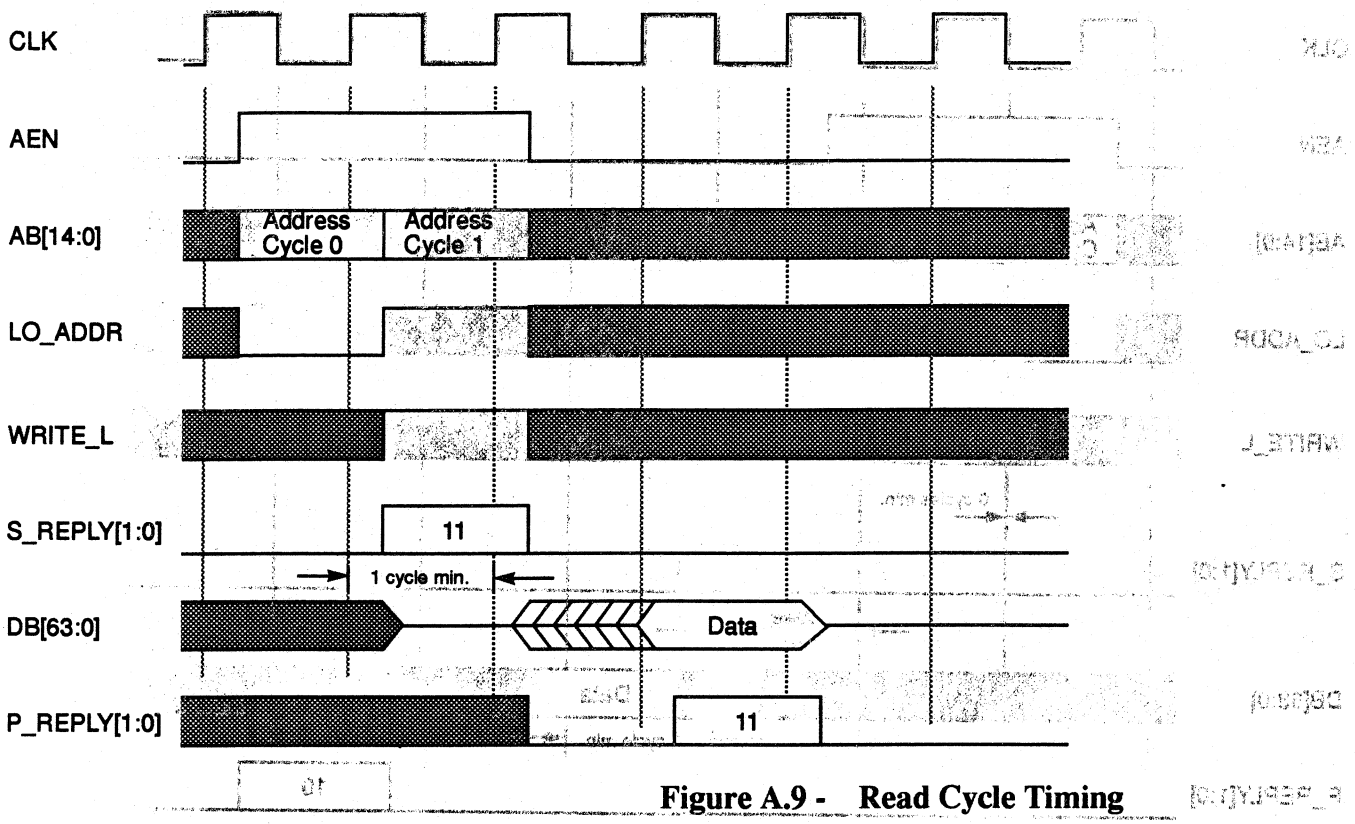


Figure A.9 - Read Cycle Timing

Back-To-Back Writes and Reads Cycles

Figure A.10 shows timing for back-to-back writes and reads. This example shows the fastest possible Controller and Slave cycles. It also assumes the read launching bit in the CPU is set to one.

The letter bubbles (A, B, C and so on) denote the pipelining of the command. A through D are write cycles. E and F are read cycles.

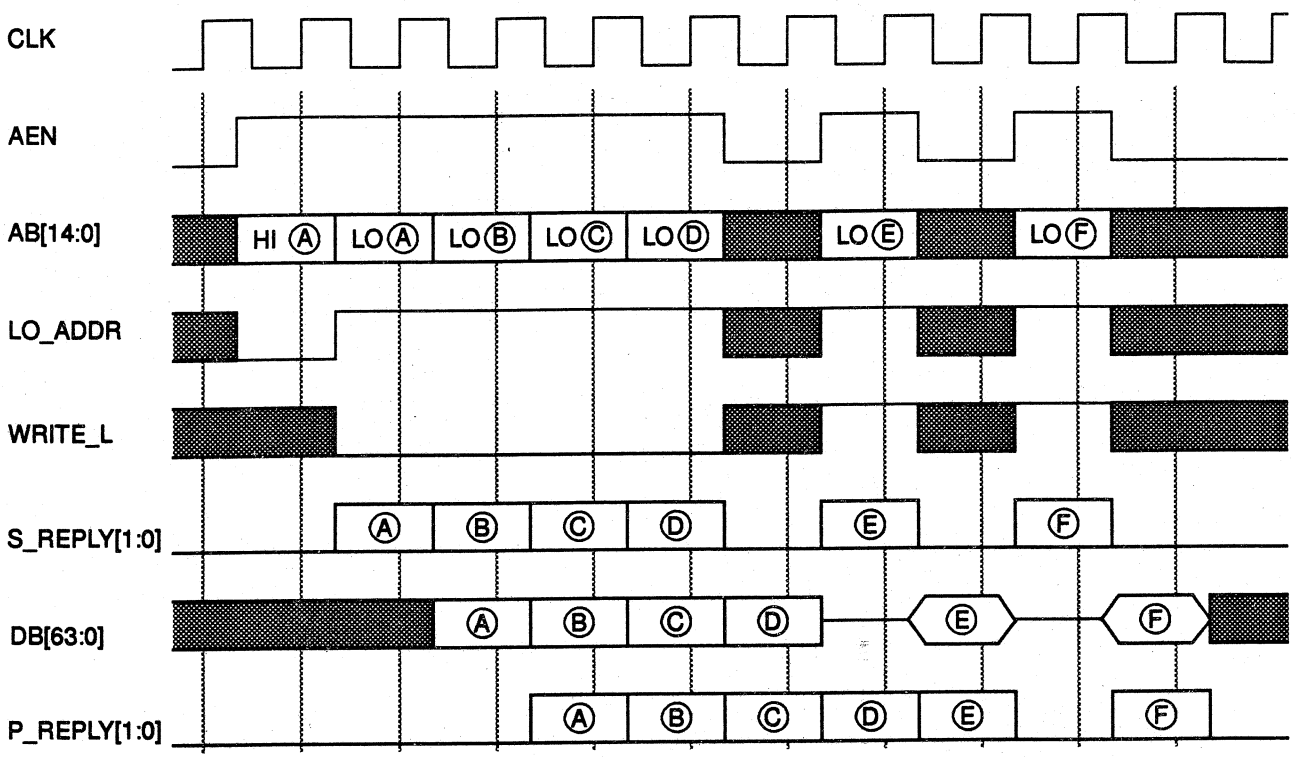


Figure A.10 - Back-To-Back Write and Read Timing